异步图书
www.epubit.com.cn

释放贝叶斯框架的灵活性与力量

Python贝叶斯分析

[阿根廷] 奥斯瓦尔多·马丁（Osvaldo Martin） 著
田俊 译

Bayesian Analysis
with Python

Packt


中国工信出版集团


人民邮电出版社
POSTS & TELECOM PRESS

Bayesian Analysis with Python

Python贝叶斯分析

[阿根廷] 奥斯瓦尔多·马丁（Osvaldo Martin） 著
田俊 译

人民邮电出版社
北京

图书在版编目（CIP）数据

Python贝叶斯分析 / （阿根廷）奥斯瓦尔多·马丁（Osvaldo Martin）著；田俊译. -- 北京：人民邮电出版社，2018.2
ISBN 978-7-115-47617-3

Ⅰ. ①P… Ⅱ. ①奥… ②田… Ⅲ. ①软件工具—程序设计②贝叶斯理论 Ⅳ. ①TP311.561②O225

中国版本图书馆CIP数据核字(2017)第324763号

版权声明



◆ 著　　　[阿根廷]奥斯瓦尔多·马丁（Osvaldo Martin）
译　　　田　俊
责任编辑　王峰松
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京捷迅佳彩印刷有限公司印刷

◆ 开本：720×960　1/16
印张：14.75
字数：241 千字　　　2018 年 2 月第 1 版
印数：1 – 3 000 册　　　2018 年 2 月北京第 1 次印刷

著作权合同登记号　图字：01-2017-3662 号

定价：69.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

内容提要

本书从务实和编程的角度讲解了贝叶斯统计中的主要概念，并介绍了如何使用流行的 PyMC3 来构建概率模型。阅读本书，读者将掌握实现、检查和扩展贝叶斯统计模型，从而提升解决一系列数据分析问题的能力。本书不要求读者有任何统计学方面的基础，但需要读者有使用 Python 编程方面的经验。

作译者简介

作者简介

Osvaldo Martin 是阿根廷国家科学与技术理事会（CONICET）的一名研究员。该理事会是负责阿根廷科技进步的主要组织。Osvaldo Martin 曾从事结构生物信息学和计算生物学方面的研究，特别是在如何验证结构蛋白质的模型方面有着深入研究；此外，在应用马尔科夫—蒙特卡洛方法模拟分子方向上有着丰富的经验，尤其喜欢用 Python 解决数据分析问题。他曾讲授结构生物信息学、Python 编程等课程，最近还开设了贝叶斯数据分析的课程。Python 和贝叶斯统计改变了他对科学的认知和对问题的思考方式。他写本书的最大动力是希望借助 Python 帮助大家理解概率模型，同时，他也是 PyMOL 社区（一个基于 C/Python 的分子可视化社区）的活跃成员，最近他也对 PyMC3 社区做了一些贡献。

我要感谢我的妻子 Romina 在我写作本书的过程中对我的支持，以及对我所有“疯狂”项目的支持。我还要感谢 Walter Lapadula、Juan Manuel Alonso 和 Romina Torres-Astorga，他们为本书提供了宝贵的反馈和建议。

此外，我还要特别感谢 PyMC3 的核心开发者们，本书正是因为他们的辛勤奉献才得以成为可能，希望本书能够促进 PyMC3 的传播和使用。

技术审校者简介

Austin Rochford 是 Monetate Labs 的首席数据科学家，他开发的产品用于帮助零售商在每年数十亿的活动中进行个性化销售。同时他还是一位积极倡导贝叶斯方法的数学家。

译者简介

田俊，计算机专业硕士。2016 年毕业于中国科学院自动化研究所，主要研究方向为自然语言处理中的短文本分类，毕业后曾在滴滴出行担任算法工程师，目前在微软从事自然语言处理方面的工作。

中文版审校者简介

劳俊鹏，心理学博士，PyMC 团队成员。2014 年毕业于英国格拉斯哥大学，主要研究认知神经心理学。2013 年至今在瑞士弗里堡大学从事心理学研究，专攻数据建模分析和神经计算模型。



前言

贝叶斯统计距今已经有超过 250 年的历史，其间该方法既饱受赞誉又备受轻视。直到近几十年，得益于理论进步和计算能力的提升，贝叶斯统计才越来越多地受到来自统计学以及其他学科乃至学术圈以外工业界的重视。现代的贝叶斯统计主要是计算统计学，人们对模型的灵活性、透明性以及统计分析结果的可解释性的追求最终造就了该趋势。

本书将从实用的角度来学习贝叶斯统计，不会过多地考虑统计学范例及其与贝叶斯统计之间的关系。本书的目的是借助 Python 做贝叶斯数据分析，尽管与之相关的哲学讨论也很有趣，不过受限于篇幅，这部分内容并不在本书的讨论范围之内，有兴趣的读者可以通过其他方式深入了解。

这里我们采用建模的方式学习统计学，学习如何从概率模型的角度思考问题，结合模型和数据利用贝叶斯理论推导出逻辑结论。这种建模方式使用的是一种数值计算的方法，其中，模型部分会由基于 PyMC3 的代码构建。PyMC3 是用于贝叶斯统计的 Python 库，它为用户封装了大量的数学细节和计算过程。贝叶斯方法在理论上源于概率论，这也是为什么许多讲贝叶斯方法的书中都充斥着需要一定数学基础的公式。学习统计学方面的数学知识显然有利于构建更好的模型，同时还能让你对问题、模型和结果有更好的直觉。不过类似 PyMC3 的库能够帮助你在有限的数学知识水平下学习并掌握贝叶斯统计。在阅读本书的过程中，你将亲自见证这一过程。

本书结构

第 1 章，**概率思维——贝叶斯推断指南**，介绍了贝叶斯理论及其在数据分析中的意义，并进一步阐述了贝叶斯思维方式的定义以及为什么和如何使用概率来处理不确定性。本章还包含本书其余章节中的一些基本概念。

第 2 章，**概率编程——PyMC3 编程指南**，从计算的角度重新回顾了前一章

提到的概念。这一章中我们将引入 PyMC3，并学习如何用它来构建概率模型、对后验进行采样、判断采样是否正确以及分析和解释贝叶斯结果。

第 3 章，**多参和分层模型**，介绍了贝叶斯模型中最基础的内容，并在此基础上加入了一些更复杂的内容。我们将学习如何利用多个参数构建并分析模型，以及如何利用分层模型的优势往模型中添加结构。

第 4 章，**利用线性回归模型理解并预测数据**，介绍了线性回归模型的广泛应用以及如何将其应用于更复杂的模型。在本章中，我们将学习如何利用线性模型解决回归问题，以及如何处理异常值和多变量的问题。

第 5 章，**利用逻辑回归对结果进行分类**，在前一章的基础上对线性模型做了进一步推广，将其应用于解决多输入 / 多输出分类问题。

第 6 章，**模型比较**，讨论了统计和机器学习中一些常见的模型比较难点。我们将学习一些信息测准和贝叶斯因子方面的理论知识，以及如何将其应用于模型比较。

第 7 章，**混合模型**，讨论了如何将简单的模型混合在一起构建出更复杂的模型，这种方法将引导我们认识新的模型，并从混合模型的角度重新回顾前面几章中学到的模型。此外，本章还讨论了如何进行数据聚类和如何处理计数类型数据等问题。

第 8 章，**高斯过程**，简要讨论了一些非参数统计方面的高级概念作为本书的结束，包括什么是核函数、如何使用线性核回归以及如何将高斯过程用于回归问题。

准备工作

本书代码部分使用的是 Python 3.5 以上版本，因此，建议你使用 Python 3 的最新版，尽管本书的大部分代码都能在更早的版本上运行（包括 Python 2.7，不过可能需要稍微修改）。

安装 Python 和 Python 库最简单的方法是使用 Anaconda（一个用于科学计算的软件），你可以通过网站 https://www.continuum.io/downloads 了解和下载 Anaconda。安装好 Anaconda 之后，可以使用 `conda install` 库的名称来安装 Python 库。

本书会用到以下 Python 库：

- Ipython 5.0；
- NumPy 1.11.1；
- SciPy 0.18.1；
- Pandas 0.18.1；
- Matplotlib 1.5.3；
- Seaborn 0.7.1；
- PyMC3 3.0。

读者对象

本书的阅读对象为不熟悉贝叶斯统计方法，同时又希望学习如何进行贝叶斯数据分析的本科生、研究生或数据科学家。本书不要求读者有统计学或贝叶斯分析方面的背景，书中尽可能地减少了数学公式的使用，只在我们认为有利于读者理解相关概念的地方用到。此外，所有的概念都通过代码、图表以及文字进行了详细描述。本书假设读者知道如何使用 Python 进行编程，最好熟悉 NumPy、Matplotlib 或者 Pandas。

惯例

在阅读本书过程中，你会看到一些不同的排版方式用于区分不同的信息，以下是这些排版的例子及其解释。

文字中的代码单词、数据表的名字、文件夹名、文件名、文件扩展名、路径、链接、用户输入都按以下方式排版：“为了准确计算 HPD，我们将使用 `plot_post` 函数”。

代码片段的排版方式如下：

```
n_params = [1, 2, 4]
p_params = [0.25, 0.5, 0.75]
x = np.arange(0, max(n_params)+1)
f, ax = plt.subplots(len(n_params), len(p_params), sharex=True, sharey=True)
```

```
    for i in range(3):
        for j in range(3):
            n = n_params[i]
            p = p_params[j]
            y = stats.binom(n=n, p=p).pmf(x)
            ax[i,j].vlines(x, 0, y, colors='b', lw=5)
            ax[i,j].set_ylim(0, 1)
            ax[i,j].plot(0, 0, label="n = {:3.2f}\np = {:3.2f}".format(n,
p), alpha=0)
            ax[i,j].legend(fontsize=12)
    ax[2,1].set_xlabel('$\\theta$', fontsize=14)
    ax[1,0].set_ylabel('$p(y|\\theta)$', fontsize=14)
    ax[0,0].set_xticks(x)
```

所有命令行的输入或输出都按以下方式排版：

conda install NamePackage

读者反馈

欢迎读者对本书的反馈，让我们知道你关于这本书的想法——喜欢什么，不喜欢什么。读者反馈对于我们很重要，它可以帮助我们开发读者真正需要的话题。想给我们发送反馈，只需要发送电子邮件至 feedback@packtpub.com，并在邮件主题中告知书名。如果你是某个话题的专家，并且有兴趣编写书籍或者给予贡献，请查看我们的作者指导：www.packtpub.com/authors 。

客户支持

你现在已经是 Packt 书籍的荣誉所有者。你还拥有以下权利。

下载示例代码

你可以从 http://www.packtpub.com 的个人账户下载本书的示例代码文件。如果是从别的地方购买的本书，可以访问 http://www.packtpub.com/support，注册后，代码文件会直接通过电子邮件发送给你。你也可以通过下列步骤下载代码文件。

（1）使用你的邮箱和密码在我们的网站登录并注册。

（2）在顶部的 SUPPORT 标签上悬停光标。

（3）单击 Code Downloads & Errata 按钮。

（4）在 Search 文本框中输入书名。

（5）选取代码文件所在的书籍。

（6）选择购书途径的下拉菜单。

（7）单击 Code Download 按钮。

你也可以在本书网站的页面单击 Code Files 按钮下载代码文件。可以通过在 Search 文本框中输入书名找到本书的网页。你需要登录自己的 Packt 账户。

文件下载完成后，确保使用下列软件的最新版解压或抽取文件：

- WinRAR / 7-Zip for Windows；
- Zipeg / iZip / UnRarX for Mac；
- 7-Zip / PeaZip for Linux。

本书涉及的代码也可以在 GitHub 上下载 https://github.com/ PacktPublishing/Bayesian-Analysis-with-Python。此外，在 https://github. com/PacktPublishing/ 中还可以查看一系列其他书籍和视频的代码。

下载本书的彩图

我们还提供了本书中所有彩色图表的 PDF 文件，从而帮助你理解印刷造成的差异。你可以从 https://www.packtpub. com/sites/default/files/downloads/BayesianAnalysiswithPython_ColorImages.pdf 下载。

勘误

尽管我们已经非常细心地努力保证内容的正确性，但是错误还是不可避免。如果你在本书中发现错误，不管是文本错误或是代码错误，请告诉我们。你的善举会省去其他用户的烦恼，并帮助我们改进本书的后续版本。如果你发现了任何错误，请访问 http://www.packtpub.com/submit-errata 报告给我们。你只需选取书

名，单击 Errata Submission Form 链接，输入勘误的具体信息。一旦勘误确定之后，我们会接受你的提交。勘误会上传到我们的网站，或者添加到书籍勘误部分已有的勘误列表下。要查看以前提交的勘误，访问 https://www.packtpub.com/books/content/support，在搜索框中输入书名，所需信息便会出现在 Errata 部分下。此外，你也可以在以下地址查看和提交勘误信息：https://github.com/aloctavodia/BAP。

版权

互联网上版权资料的盗版问题一直是所有媒介关心的问题。在 Packt，我们一直严肃对待版权和许可的保护问题。如果你在互联网上遇到任何形式的我社出版物的非法副本，请立即把具体地址或者网站名称提供给我们，请联系 copyright@packtpub.com，附上可疑的盗版材料的链接。我们非常感谢你在保护作者方面所做的努力，我们会注重提升自我能力，给你带来更有价值的内容。

疑问

如果你对本书有任何疑问，可以联系我们：questions@packtpub.com，我们会尽全力解决你的问题。

目录


人民邮电
出版社


人民邮电
出版社

第 1 章
概率思维——贝叶斯推断指南

归根到底，概率论不过是把常识化作计算而已。

——皮埃尔—西蒙·拉普拉斯

本章我们将学习贝叶斯统计中的核心概念以及一些用于贝叶斯分析的基本工具。大部分内容都是一些理论介绍，其中会涉及一些 Python 代码，绝大多数概念会在本书其余章节中反复提到。尽管本章内容有点偏理论，可能会让习惯代码的你感到有点不安，不过这会让你在后面应用贝叶斯统计方法解决问题时容易一些。

本章包含以下主题：

- 统计模型；
- 概率及不确定性；
- 贝叶斯理论及统计推断；
- 单参数推断以及经典的抛硬币问题；
- 如何选择先验；
- 如何报告贝叶斯分析结果；
- 安装所有相关的 Python 库。

1.1　以建模为中心的统计学

统计学主要是收集、组织、分析并解释数据，因此，统计学的基础知识对数据分析来说至关重要。分析数据时一个非常有用的技巧是知道如何运用某种编程语言（如 Python）编写代码。真实世界里充斥着复杂而杂乱的数据，因此对数据做一些预处理操作必不可少。即便你的数据已经是整理好的了，掌握一定的编程技巧仍然会给你带来很大帮助，因为如今的贝叶斯统计绝大多数都是

计算统计学。

大多数统计学导论课程（对非统计学专业的人而言）一般就像展示一本菜谱书，每一种统计方法就是一个菜谱：首先，到统计学的后厨取出一个罐头打开，放点数据上去尝尝，然后不停搅拌直到得出一个稳定的 p 值，该值最好低于 0.05（如果你不知道什么是 p 值，别担心，本书不会涉及这些概念）。这类课程的目的是教会你如何选择一个合适的罐头。本书采用的是另外一种方式：首先我们也需要点原料，不过这次是自己亲自做的而不是买来的罐头，然后学习如何把新鲜的食材混合在一起以适应不同的烹饪场景。在正式烹饪之前，我们先学点统计学的术语和概念。

1.1.1　探索式数据分析

数据是统计学最基本的组成部分。数据的来源多，比如实验、计算机模拟、调查以及观测等。假如我们是数据生成或收集人员，首先要考虑的是要解决什么样的问题以及打算采用什么方法，然后再去着手准备数据。事实上，统计学有一个叫做**实验设计**的分支专门研究如何获取数据。在这个数据泛滥的年代，我们有时候会忘了获取数据并非总是很便宜。例如，尽管大型强子碰撞加速装置一天能产生上百 TB 的数据，但其建造却要花费数年的人力和智力。本书假设我们已经获取了数据并且数据是整理好的（这在现实中通常很少见），以便关注到本书的主题上来。如果你想学习如何用 Python 做数据清洗和分析并进一步学习机器学习，你可以阅读 Jake VanderPlas 写的《Python Data Science Handbook》一书。

假设我们已经有了数据集，通常的做法是先对其探索并可视化，这样我们就能对手头的数据有个直观的认识。可以通过如下两步完成所谓的**探索式数据分析**过程：

- 描述性统计；
- 数据可视化。

其中，描述性统计是指如何用一些指标或统计值来定量地总结或刻画数据，例如你已经知道了如何用均值、众数、标准差、四分位差等指标来描述数据。数

据可视化是指用生动形象的方式表述数据，你大概对直方图、散点图等表现形式比较熟悉。乍看起来，探索式数据分析似乎是在复杂分析之前的一些准备工作，或者是作为一些复杂分析方法的替代品，不过探索式数据分析在理解、解释、检查、总结及交流贝叶斯分析结果等过程中依然有用。

1.1.2 统计推断

有时候，画画图、对数据做些简单的计算（比如求均值）就够了。另外一些时候，我们希望从数据中挖掘出一些更一般性的结论。我们可能希望了解数据是怎么生成的，也可能是想对未来还未观测到的数据做出预测，又或者是希望从多个对观测值的解释中找出最合理的一个，这些正是统计推断所做的事情。模型分为许多种，统计推断依赖的是概率模型，许多科学研究（以及我们对真实世界的认识）也都是基于模型的，大脑不过是对现实进行建模的一台机器，可以观看相关的 TED 演讲了解大脑是如何对现实进行建模的，网址为 http://www.tedxriodelaplata.org/videos/m%C3%A1quina-construye-realidad。

什么是模型？模型是对给定系统或过程的一种简化描述。这些描述只关注系统中某些重要的部分，因此，大多数模型的目的并不是解释整个系统。此外，假如我们有两个模型能用来解释同一份数据并且效果差不多，其中一个简单点，另一个复杂一些，通常我们倾向于更简单的模型，这称作**奥卡姆剃刀**，我们会在第 6 章模型比较部分讨论贝叶斯分析与其之间的联系。

不管你打算构建哪种模型，模型构建都遵循一些相似的基本准则，我们把贝叶斯模型的构建过程总结为如下 3 步。

（1）给定一些数据以及这些数据是如何生成的假设，然后构建模型。通常，这里的模型都是一些很粗略的近似，不过大多时候也够用了。

（2）利用贝叶斯理论将数据和模型结合起来，根据数据和假设推导出逻辑结论，我们称之为经数据拟合后的模型。

（3）根据多种标准，包括真实数据和对研究问题的先验知识，判断模型拟合得是否合理。

通常，我们会发现实际的建模过程并非严格按照该顺序进行的，有时候我们

有可能跳到其中任何一步，原因可能是编写的程序出错了，也可能是找到了某种改进模型的方式，又或者是我们需要增加更多的数据。

贝叶斯模型是基于概率构建的，因此也称作概率模型。为什么基于概率呢？因为概率这个数学工具能够很好地描述数据中的不确定性，接下来我们将对其进行深入了解。

1.2　概率与不确定性

尽管概率论是数学中一个相当成熟和完善的分支，但关于概率的诠释仍然有不止一种。对于贝叶斯派而言，概率是对某一命题不确定性的衡量。假设我们对硬币一无所知，同时没有与抛硬币相关的任何数据，那么可以认为正面朝上的概率介于 0 到 1 之间，也就是说，在缺少信息的情况下，所有情况都是有可能发生的，此时不确定性也最大。假设现在我们知道硬币是公平的，那么我们可以认为正面朝上的概率是 0.5 或者是 0.5 附近的某个值（假如硬币不是绝对公平的话），如果此时收集数据，我们可以根据观测值进一步更新前面的先验假设，从而降低对该硬币偏差的不确定性。按照这种定义，提出以下问题都是自然而且合理的：火星上有多大可能存在生命？电子的质量是 9.1×10^{-31}kg 的概率是多大？ 1816 年 7 月 9 号是晴天的概率是多少？值得注意的是，类似火星上是否有生命这种问题的答案是二值化的，但我们关心的是，基于现有数据以及我们对火星物理条件和生物条件的了解，火星上存在生命的概率有多大。该命题取决于我们当前所掌握的信息而非客观的自然属性。我们使用概率是因为我们对事件不确定，而不代表事件本身是不确定的。由于这种概率的定义取决于我们的认知水平，有时也被称为概率的主观定义，这也就解释了为什么贝叶斯派总被称作主观统计。然而，这种定义并不是说所有命题都是同等有意义的，仅是承认我们对世界的理解是基于现有的数据和模型，是不完备的。不基于模型或理论去理解世界是不可能的。即使我们能脱离社会预设，最终也会受到生物上的限制：受进化过程影响，我们的大脑已经与世界上的模型建立了联系。我们注定要以人类的方式去思考，永远不可能像蝙蝠或其他动物那样思考。而且，宇宙是不确定的，总的来说我们能做的就是对其进行概率性描述。需要注意的是，不管世界的本原是确定的还是随机的，我们都将概率作为衡量不确定性的工具。

逻辑是指如何有效地推论，在亚里士多德学派或者经典的逻辑学中，一个命题只能是对或者错，而在贝叶斯学派定义的概率中，确定性只是概率上的一种特殊情况：一个正确命题的概率值为 1，而一个错误命题的概率值为 0。只有在我们拥有充分的数据表明火星上存在生物生长和繁殖时，我们才认为“火星上存在生命”这一命题为真的概率值为 1。不过，需要注意的是，认定一个命题的概率值为 0 通常是比较困难的，这是因为火星上可能存在某些地方还没被探测到，或者是我们的实验有问题，又或者是某些其他原因导致我们错误地认为火星上没有生命而实际上有。与此相关的是克伦威尔准则（Cromwell’s Rule），其含义是指在对逻辑上正确或错误的命题赋予概率值时，应当避免使用 0 或者 1。有意思的是，考克斯（Cox）在数学上证明了如果想在逻辑推理中加入不确定性，我们就必须使用概率论的知识，由此来说，贝叶斯定理可以视为概率论逻辑上的结果。因此，从另一个角度来看，贝叶斯统计是对逻辑学处理不确定性问题的一种扩充，当然这里毫无对主观推理的轻蔑。现在我们已经熟悉了贝叶斯学派对概率的理解，接下来就了解下概率相关的数学特性吧。如果你想深入学习概率论，可以参考阅读 Joseph K Blitzstein 和 Jessica Hwang 写的《概率导论》（*Introduction to Probability*）。

概率值介于 [0,1] 之间（包括 0 和 1），其计算遵循一些法则，其中之一是乘法法则：

$$p(A,B) = p(A \mid B)\,p(B)$$

上式中，A 和 B 同时发生的概率值等于 B 发生的概率值乘以在 B 发生的条件下 A 也发生的概率值，其中，$p(A,B)$表示 A 和 B 的联合概率，$p(A \mid B)$表示条件概率，二者的现实意义是不同的，例如，路面是湿的概率跟下雨时候路面是湿的概率是不同的。条件概率可能比原来的概率高，也可能低。如果 B 并不能提供任何关于 A 的信息，那么，$p(A \mid B) = p(A)$，也就是说，A 和 B 是相互独立的。相反，如果事件 B 能够给出关于事件 A 的一些信息，那么根据事件 B 提供的信息不同，事件 A 可能发生的概率会变得更高或是更低。

条件概率是统计学中的一个核心概念，接下来我们将看到，理解条件概率对于理解贝叶斯理论至关重要。这里我们换个角度来看条件概率，假如我们把前面的公式调整下顺序，就可以得到下面的公式：

$$p(A \mid B)=\frac{p(A,B)}{p(B)}$$

需要注意的是，我们不对 0 概率事件计算条件概率，因此，分母的取值范围是 (0,1)，从而可以看出条件概率大于或等于联合概率。为什么要除以 $p(B)$ 呢？因为在已经知道事件 B 发生的条件下，我们考虑的可能性空间就缩小到了事件 B 发生的范围内，然后将该范围内 A 发生的可能性除以 B 发生的可能性便得到了条件概率 $p(A \mid B)$。需要强调的是，所有的概率本质上都是条件概率，世间并没有绝对的概率。不管我们是否留意，在谈到概率时总是存在这样或那样的模型、假设或条件。比如，当我们讨论明天下雨的概率时，在地球上、火星上甚至宇宙中其他地方明天下雨的概率是不同的。同样，硬币正面朝上的概率取决于我们对硬币有偏性的假设。在理解概率的含义之后，接下来我们讨论另一个话题：概率分布。

1.2.1　概率分布

概率分布是数学中的一个概念，用来描述不同事件发生的可能性，通常这些事件限定在一个集合内，代表了所有可能发生的事件。在统计学里可以这么理解：数据是从某种参数未知的概率分布中生成的。由于并不知道具体的参数，我们只能借用贝叶斯定理从仅有的数据中反推参数。概率分布是构建贝叶斯模型的基础，不同分布组合在一起之后可以得到一些很有用的复杂模型。

本书会介绍一些概率分布，在第一次介绍某个概率分布时，我们会先花点时间理解它。最常见的一种概率分布是高斯分布，又叫正态分布，其数学公式描述如下：

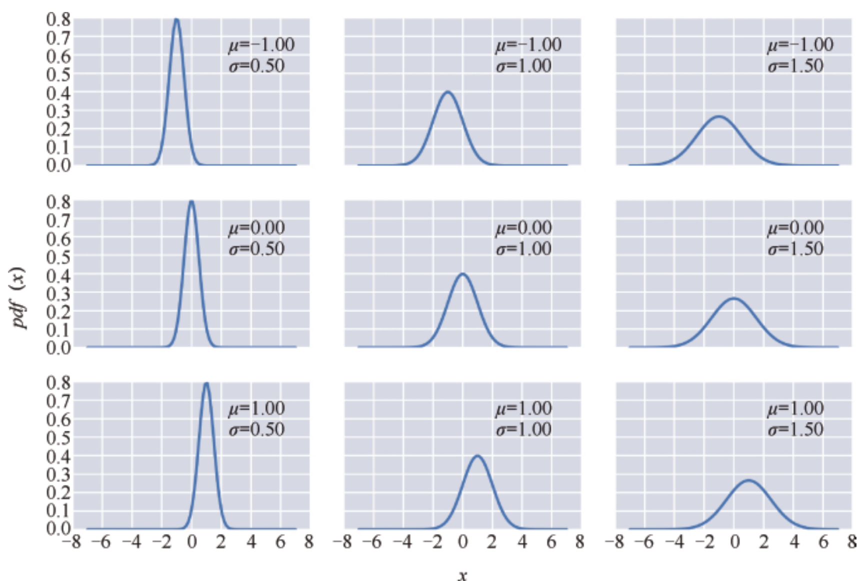
$$pdf(x \mid \mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}\mathrm{e}^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

上式中，μ 和 σ 是高斯分布的两个参数。第 1 个参数 μ 是该分布的均值（同时也是中位数和众数），其取值范围是任意实数，即$\mu \in \mathbb{R}$；第 2 个参数 σ 是标准差，用来衡量分布的离散程度，其取值只能为正。由于μ和σ的取值范围无穷大，因此高斯分布的实例也有无穷多。虽然数学公式这一表达形式简洁明了，也有人称之有美感，不过得承认公式还是有些不够直观，我们可以尝试用 Python 代码将公式的含义重新表示出来。首先看看高斯分布都长什么样：

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
import seaborn as sns

mu_params = [-1, 0, 1]
sd_params = [0.5, 1, 1.5]
x = np.linspace(-7, 7, 100)
f, ax = plt.subplots(len(mu_params), len(sd_params), sharex=True, sharey=True)
for i in range(3):
    for j in range(3):
        mu = mu_params[i]
        sd = sd_params[j]
        y = stats.norm(mu, sd).pdf(x)
        ax[i,j].plot(x, y)
        ax[i,j].plot(0, 0,
        label="$\\mu$ = {:3.2f}\n$\\sigma$ = {:3.2f}".format (mu, sd), alpha=0)
        ax[i,j].legend(fontsize=12)
ax[2,1].set_xlabel('$x$', fontsize=16)
ax[1,0].set_ylabel('$pdf(x)$', fontsize=16)
plt.tight_layout()
```

上面代码的输出结果如下：



由概率分布生成的变量（例如 x）称作**随机变量**，当然这并不是说该变量可以取任意值，相反，我们观测到该变量的数值受到概率分布的约束，而其随机性

源于我们只知道变量的分布却无法准确预测该变量的值。通常，如果一个随机变量服从在参数 μ 和 σ 下的高斯分布，我们可以这样表示该变量：

$$x \sim N(\mu, \sigma)$$

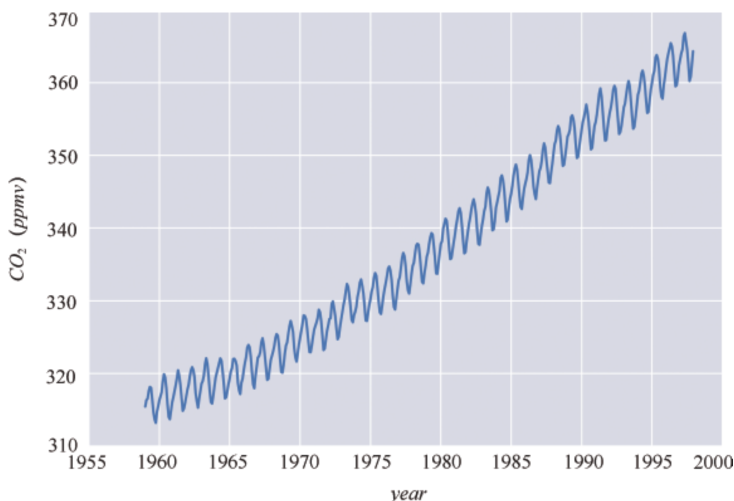
其中，符号～读作服从于某种分布。

随机变量分为两种：**连续变量**和**离散变量**。连续随机变量可以从某个区间内取任意值（我们可以用 Python 中的浮点型数据来表示），而离散随机变量只能取某些特定的值（我们可以用 Python 中的整型数据来表示）。

许多模型都假设，如果对服从于同一个分布的多个随机变量进行连续采样，那么各个变量的采样值之间相互独立，我们称这些随机变量是独立同分布的。用数学语言描述就是，如果两个随机变量 x 和 y 对于所有可能的取值都满足 $p(x,y) = p(x)p(y)$，那么称这两个变量相互独立。

时间序列是不满足独立同分布的一个典型例子。在时间序列中，需要对时间维度的变量多加留心。下面的例子是从 http://cdiac.esd.ornl.gov 中获取的数据。这份数据记录了从 1959 年到 1997 年大气中二氧化碳的含量。我们用以下代码将它写出来：

```
data = np.genfromtxt('mauna_loa_CO2.csv', delimiter=',')
plt.plot(data[:,0], data[:,1])
plt.xlabel('$year$', fontsize=16)
plt.ylabel('$CO_2 (ppmv)$', fontsize=16)
```



图中每个点表示每个月空气中二氧化碳含量的测量值，可以看到测量值是与时间相关的。我们可以观察到两个趋势：一个是季节性的波动趋势（这与植物周期性生长和衰败有关）；另一个是二氧化碳含量整体性的上升趋势。

1.2.2 贝叶斯定理与统计推断

到目前为止，我们已经学习了一些统计学中的基本概念和词汇，接下来让我们首先看看神奇的贝叶斯定理：

$$p(H \mid D)=\frac{p(D \mid H)p(H)}{p(D)}$$

看起来稀松平常，似乎跟小学课本里的公式差不多，不过这就是关于贝叶斯统计你所需要掌握的全部。首先看看贝叶斯定理是怎么来的，这对我们理解它会很有帮助。事实上，我们已经掌握了如何推导它所需要的全部概率论知识。

- 根据前面提到的概率论中的乘法准则，我们有以下式子：

$$p(H,D)=p(H \mid D)p(D)$$

- 上式还可以写成如下形式：

$$p(H,D)=p(D \mid H)p(H)$$

- 由于以上式子的左边相等，于是可以得到：

$$p\left(D \mid H\right)p\left(H\right)=p\left(H \mid D\right)p\left(D\right)$$

- 对上式调整下顺序，便得到了贝叶斯定理：

$$p\left(H \mid D\right)=\frac{p\left(D \mid H\right)p\left(H\right)}{p\left(D\right)}$$

现在，让我们看看这个式子的含义及其重要性。首先，上式表明$p(D \mid H)$和$p(H \mid D)$并不一定相等，这一点非常重要，日常分析中即使系统学习过统计学和概率论的人也很容易忽略这点。我们举个简单例子来说明为什么二者不一定相等：有两条腿的动物就是人的概率和人有两条腿的概率显然是不同的。几乎所有人都有两条腿（除了某些人因为先天性原因或者意外导致没有两条腿），但是有

两条腿的动物中很多都不是人类，比如鸟类。

在前面的式子中，如果我们将 H 理解为假设，D 理解为数据，那么贝叶斯定理告诉我们的就是，在给定数据的条件下如何计算假设成立的概率。不过，如何把假设融入贝叶斯定理中去呢？答案是概率分布。换句话说，H 是一种狭义上的假设，我们所做的实际上是寻找模型的参数（更准确地说是参数的分布）。因此，与其称 H 为假设，不如称之为模型，这样能避免歧义。

贝叶斯定理非常重要，后面会反复用到，这里我们先熟悉下其各个部分的名称：

- $p(H)$：先验；
- $p(D \mid H)$：似然；
- $p(H \mid D)$：后验；
- $p(D)$：证据。

先验分布反映的是在观测到数据之前我们对参数的了解，如果我们对参数一无所知（就跟《权力的游戏》中的雪诺一样），那么可以用一个不包含太多信息的均匀分布来表示。由于引入了先验，有些人会认为贝叶斯统计是偏主观的，然而，这些先验不过是构建模型时的一些假设罢了，其主观性跟似然差不多。

似然是指如何在实验分析中引入观测数据，反映的是在给定参数下得到某组观测数据的可信度。

后验分布是贝叶斯分析的结果，反映的是在给定数据和模型的条件下我们对问题的全部认知。需要注意，后验指的是我们模型中参数的概率分布而不是某个值，该分布正比于先验乘以似然。有这么个笑话：贝叶斯学派就像是这样一类人，心里隐约期待着一匹马，偶然间瞥见了一头驴，结果坚信他看到的是一头骡子。当然了，如果要刻意纠正这个笑话的话，在先验和似然都比较含糊的情况下，我们会得到一个（模糊的）“骡子”后验。不过，这个笑话也讲出了这样一个道理，后验其实是对先验和似然的某种折中。从概念上讲，后验可以看做是在观测到数据之后对先验的更新。事实上，一次分析中的后验，在收集到新的数据之后，也可以看做是下一次分析中的先验。这使得贝叶斯分析特别适合于序列化的数据分析，比如通过实时处理来自气象站和卫星的数据从而提前预警灾害，更

详细的内容可以阅读**在线机器学习**方面的算法。

最后一个概念是**证据**，也称作边缘似然。正式地讲，证据是在模型的参数取遍所有可能值的条件下得到指定观测值的概率的平均。不过，本书的大部分内容并不关心这个概念，我们可以简单地把它当作归一化系数。我们只关心参数的相对值而非绝对值。把证据这一项忽略掉之后，贝叶斯定理可以表示成如下正比例形式：

$$p(H \mid D) \propto p(D \mid H) p(H)$$

理解其中的每个概念可能需要时间和更多的例子，本书也将围绕这些内容展开。

1.3 单参数推断

前面，我们学习了几个重要概念，其中有两个是贝叶斯统计的核心概念，这里我们用一句话再重新强调下：概率是用来衡量参数不确定性的，贝叶斯定理就是用来在观测到新的数据时正确更新这些概率以期降低我们的不确定性。

现在我们已经知道什么是贝叶斯统计了，接下来就从一个简单的例子入手，通过推断单个未知参数来学习如何进行贝叶斯统计。

1.3.1 抛硬币问题

抛硬币是统计学中的一个经典问题，其描述如下：我们随机抛一枚硬币，重复一定次数，记录正面朝上和反面朝上的次数，根据这些数据，我们需要回答诸如这枚硬币是否公平，以及更进一步这枚硬币有多不公平等问题。抛硬币是一个学习贝叶斯统计非常好的例子，一方面是因为几乎人人都熟悉抛硬币这一过程，另一方面是因为这个模型很简单，我们可以很容易计算并解决这个问题。此外，许多真实问题都包含两个互斥的结果，例如 0 或者 1、正或者负、奇数或者偶数、垃圾邮件或者正常邮件、安全或者不安全、健康或者不健康等。因此，即便我们讨论的是硬币，该模型也同样适用于前面这些问题。

为了估计硬币的偏差，或者更广泛地说，想要用贝叶斯学派理论解决问题，我们需要数据和一个概率模型。对于抛硬币这个问题，假设我们已试验了

一定次数并且记录了正面朝上的次数，也就是说数据部分已经准备好了，剩下的就是模型部分了。考虑到这是第一个模型，我们会列出所有必要的数学公式，并且一步一步推导。下一章中，我们会重新回顾这个问题，并借用 PyMC3 从数值上解决它（也就是说那部分不需要手动推导，而是利用 PyMC3 和计算机来完成）。

通用模型

首先，我们要抽象出偏差的概念。我们称，如果一枚硬币总是正面朝上，那么它的偏差就是 1，反之，如果总是反面朝上，那么它的偏差就是 0，如果正面朝上和反面朝上的次数各占一半，那么它的偏差就是 0.5。这里用参数 θ 来表示偏差，用 y 表示 N 次抛硬币实验中正面朝上的次数。根据贝叶斯定理，我们有如下公式：

$$p(\theta \mid y) \propto p(y \mid \theta) p(\theta)$$

这里需要指定我们将要使用的先验 $p(\theta)$ 和似然 $p(y \mid \theta)$ 分别是什么。让我们首先从似然开始。

选择似然

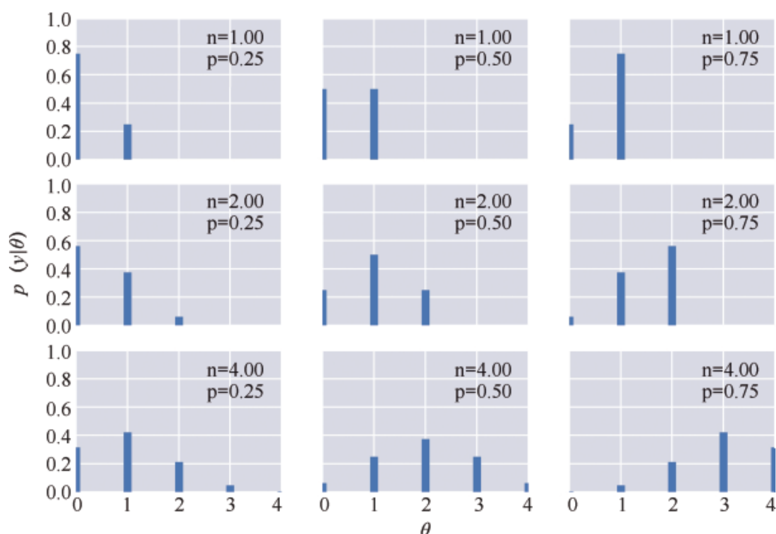
假设多次抛硬币的结果相互之间没有影响，也就是说每次抛硬币都是相互独立的，同时还假设结果只有两种可能：正面朝上或者反面朝上。但愿你能认同我们对这个问题做出的合理假设。基于这些假设，一个不错的似然候选是二项分布：

$$p(y \mid \theta) = \frac{N!}{y!(N-y)!} \theta^{y} (1-\theta)^{(N-y)}$$

这是一个离散分布，表示 N 次抛硬币实验中 y 次正面朝上的概率（或者更通俗地描述是，N 次实验中，y 次成功的概率）。下面的代码生成了 9 个二项分布，每个子图中的标签显示了对应的参数：

```
n_params = [1, 2, 4]
p_params = [0.25, 0.5, 0.75]
x = np.arange(0, max(n_params)+1)
f, ax = plt.subplots(len(n_params), len(p_params), sharex=True, sharey=True)
```

```
for i in range(3):
    for j in range(3):
        n = n_params[i]
        p = p_params[j]
        y = stats.binom(n=n, p=p).pmf(x)
        ax[i,j].vlines(x, 0, y, colors='b', lw=5)
        ax[i,j].set_ylim(0, 1)
        ax[i,j].plot(0, 0, label="n = {:3.2f}\np =
         {:3.2f}".format(n, p), alpha=0)
        ax[i,j].legend(fontsize=12)
ax[2,1].set_xlabel('$\\theta$', fontsize=14)
ax[1,0].set_ylabel('$p(y|\\theta)$', fontsize=14)
ax[0,0].set_xticks(x)
```



二项分布是似然的一个合理选择，直观上讲，θ 可以看作抛一次硬币时正面朝上的可能性，并且该过程发生了 y 次。类似地，我们可以把“$1-\theta$”看作抛一次硬币时反面朝上的概率，并且该过程发生了“$N-y$”次。

假如我们知道了 θ，那么就可以从二项分布得出硬币正面朝上的分布。如果我们不知道 θ，也别灰心，在贝叶斯统计中，当我们不知道某个参数的时候，就对其赋予一个先验。接下来继续选择先验。

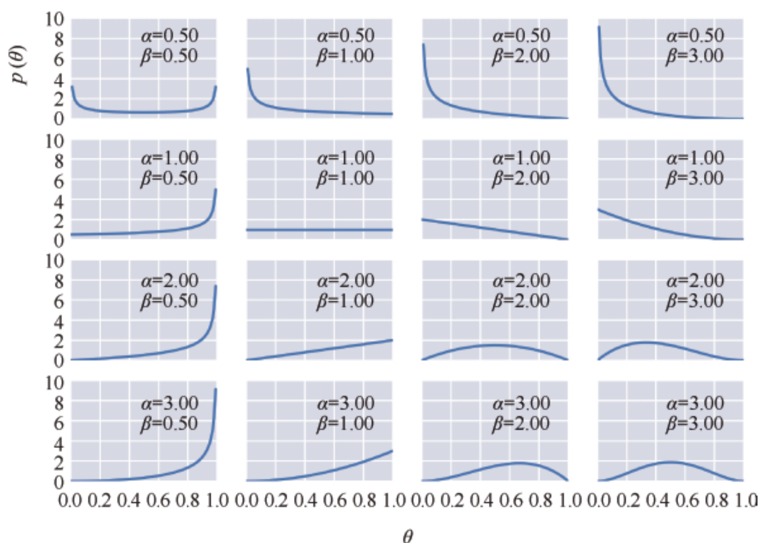
选择先验

这里我们选用贝叶斯统计中最常见的 **beta 分布**，作为先验，其数学形式如下：

$$p(\theta)=\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}$$

仔细观察上面的式子可以看出，除了 Γ 部分之外，beta 分布和二项分布看起来很像。Γ 是希腊字母中大写的伽马，用来表示伽马函数。现在我们只需要知道，用分数表示的第一项是一个正则化常量，用来保证该分布的积分为 1，此外，α 和 β 两个参数用来控制具体的分布形态。beta 分布是我们到目前为止见到的第 3 个分布，利用下面的代码，我们可以深入了解其形态：

```
params = [0.5, 1, 2, 3]
x = np.linspace(0, 1, 100)
f, ax = plt.subplots(len(params), len(params), sharex=True, sharey=True)
for i in range(4):
    for j in range(4):
        a = params[i]
        b = params[j]
        y = stats.beta(a, b).pdf(x)
        ax[i,j].plot(x, y)
        ax[i,j].plot(0, 0, label="$\\alpha$ = {:3.2f}\n$\\beta$ = {:3.2f}".format(a, b), alpha=0)
        ax[i,j].legend(fontsize=12)
ax[3,0].set_xlabel('$\\theta$', fontsize=14)
ax[0,0].set_ylabel('$p(\\theta)$', fontsize=14)
plt.savefig('B04958_01_04.png', dpi=300, figsize=(5.5, 5.5))
```

为什么要在模型中使用 beta 分布呢？在抛硬币以及一些其他问题中使用 beta 分布的原因之一是：beta 分布的范围限制在 0 到 1 之间，这跟我们的参数 θ 一样；另一个原因是其通用性，从前面的图可以看出，该分布可以有多种形状，包括均匀分布、类高斯分布、U 型分布等。第 3 个原因是：beta 分布是二项分布（前面我们使用了该分布描述似然）的共轭先验。似然的共轭先验是指，将该先验分布与似然组合在一起之后，得到的后验分布与先验分布的表达式形式仍然是一样的。简单说，就是每次用 beta 分布作为先验、二项分布作为似然时，我们会得到一个 beta 分布的后验。除 beta 分布之外还有许多其他共轭先验，例如高斯分布，其共轭先验就是自己。关于共轭先验更详细的内容可以查看 https://en.wikipedia.org/wiki/Conjugate_prior。许多年来，贝叶斯分析都限制在共轭先验范围内，这主要是因为共轭能让后验在数学上变得更容易处理，要知道贝叶斯统计中一个常见问题的后验都很难从分析的角度去解决。在建立合适的计算方法来解决任意后验之前，这只是个折中的办法。从下一章开始，我们将学习如何使用现代的计算方法来解决贝叶斯问题而不必考虑是否使用共轭先验。

计算后验

首先回忆一下贝叶斯定理：后验正比于似然乘以先验。

$$p(\theta \mid y) \propto p(y \mid \theta) p(\theta)$$

对于我们的问题而言，需要将二项分布乘以 beta 分布：

$$p(\theta \mid y) \propto \frac{N!}{y!(N-y)!} \theta^{y} (1-\theta)^{N-y} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1} (1-\theta)^{\beta-1}$$

现在，对上式进行简化。针对我们的实际问题，可以先把与 θ 不相关的项去掉而不影响结果，于是得到下式：

$$p(\theta \mid y) \propto \theta^{y} (1-\theta)^{N-y} \theta^{\alpha-1} (1-\theta)^{\beta-1}$$

重新整理之后得到：

$$p(\theta \mid y) \propto \theta^{\alpha-1+y} (1-\theta)^{\beta-1+N-y}$$

可以看出，上式和 beta 分布的形式很像（除了归一化部分），其对应的参数

分别为 $\alpha_{\text{posterior}} = \alpha_{\text{prior}} + y$、$\beta_{\text{posterior}} = \beta_{\text{prior}} + N - y$。也就是说，在抛硬币这个问题中，后验分布是如下 beta 分布：

$$p(\theta \mid y) = Beta\left(\alpha_{\text{prior}} + y, \beta_{\text{prior}} + N - y\right)$$

计算后验并画图

现在已经有了后验的表达式，我们可以用 Python 对其计算并画出结果。下面的代码中，其实只有一行是用来计算后验结果的，其余的代码都是用来画图的：

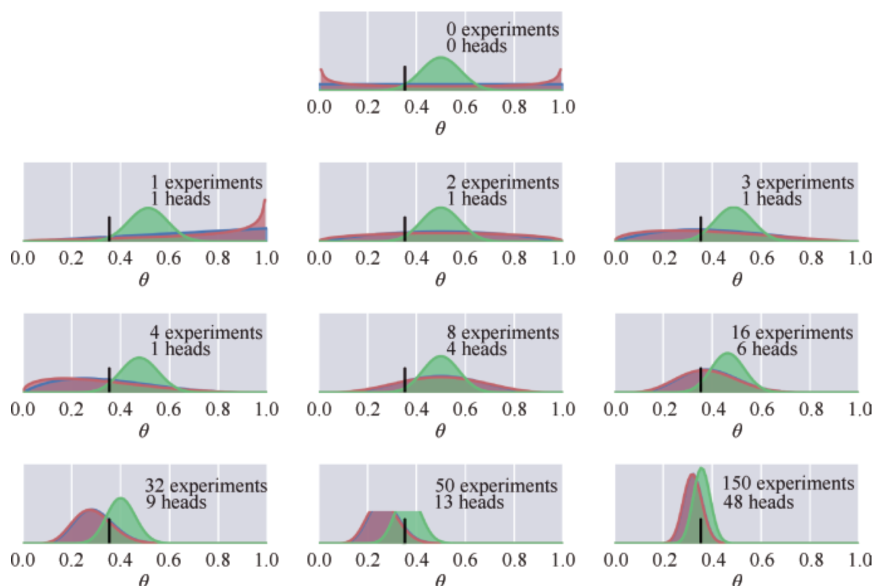
```
theta_real = 0.35
trials = [0, 1, 2, 3, 4, 8, 16, 32, 50, 150]
data = [0, 1, 1, 1, 1, 4, 6, 9, 13, 48]

beta_params = [(1, 1), (0.5, 0.5), (20, 20)]
dist = stats.beta
x = np.linspace(0, 1, 100)

for idx, N in enumerate(trials):
    if idx == 0:
        plt.subplot(4,3, 2)
    else:
        plt.subplot(4,3, idx+3)
    y = data[idx]
    for (a_prior, b_prior), c in zip(beta_params, ('b', 'r', 'g')):
        p_theta_given_y = dist.pdf(x, a_prior + y, b_prior + N - y)
        plt.plot(x, p_theta_given_y, c)
        plt.fill_between(x, 0, p_theta_given_y, color=c, alpha=0.6)

    plt.axvline(theta_real, ymax=0.3, color='k')
    plt.plot(0, 0, label="{:d} experiments\n{:d} heads".format(N, y), alpha=0)
    plt.xlim(0,1)
    plt.ylim(0,12)
    plt.xlabel(r"$\theta$")
    plt.legend()
    plt.gca().axes.get_yaxis().set_visible(False)
plt.tight_layout()
plt.savefig('B04958_01_05.png', dpi=300, figsize=(5.5, 5.5))
```





在上图的第一行中，实验的次数为 0，因此第一个图中的曲线描绘的是先验分布，其中有 3 条曲线，每条曲线分别表示一种先验。

- 蓝色的线是一个均匀分布先验，其含义是：偏差的所有可能取值都是等概率的。
- 红色的线与均匀分布有点类似，对抛硬币这个例子而言可以理解为：偏差等于 0 或者 1 的概率要比其他值更大一些。
- 最后一条绿色的线集中在中间值 0.5 附近，该分布反映了通常硬币正面朝上和反面朝上的概率大致是差不多的。我们也可以称，该先验与大多数硬币都是公平的这一信念是兼容的。“兼容”这个词在贝叶斯相关的讨论中会经常用到，特别是在提及受到数据启发的模型时。

剩余的子图描绘了后续实验的后验分布 $p(\theta|y)$，回想一下，后验可以看做是在给定数据之后更新了的先验。实验（抛硬币）的次数和正面朝上的次数分别标注在每个子图中。此外每个子图中在横轴 0.35 附近还有一个黑色的竖线，表示的是真实的 θ，显然，在真实情况下，我们并不知道该值，在这里标识出来只是为了方便理解。从这幅图中可以学到很多贝叶斯分析方面的知识。

- 贝叶斯分析的结果是后验分布而不是某个值，该分布描述了根据给定数

据和模型得到的不同数值的可能性。

- 后验最可能的值是根据后验分布的形态决定的（也就是后验分布的峰值）。
- 后验分布的离散程度与我们对参数的不确定性相关；分布越离散，不确定性越大。
- 尽管 1/2=4/8=0.5，但从图中可以看出，前者的不确定性要比后者大。这是因为我们有了更多的数据来支撑我们的推断，该直觉也同时反映在了后验分布上。
- 在给定足够多的数据时，两个或多个不同先验的贝叶斯模型会趋近于收敛到相同的结果。在极限情况下，如果有无限多的数据，不论我们使用的是怎样的先验，最终都会得到相同的后验。注意这里说的无限多是指某种程度而非某个具体的数量，也就是说，从实际的角度来讲，某些情况下无限多的数据可以通过比较少量的数据近似。
- 不同后验收敛到相同分布的速度取决于数据和模型。从前面的图中可以看出，蓝色和红色的后验在经过 8 次实验之后就很难看出区别了，而红色的曲线则一直到 150 次实验之后才与另外两个后验看起来比较接近。
- 有一点从前面的图中不太容易看出来，如果我们一步一步地更新后验，最后得到的结果跟一次性计算得到的结果是一样的。换句话说，我们可以对后验分 150 次计算，每次增加一个新的观测数据并将得到的后验作为下一次计算的先验，也可以在得到 150 次抛硬币的结果之后一次性计算出后验，而这两种计算方式得到的结果是完全一样的。这个特点非常有意义，在许多数据分析的问题中，每当我们得到新的数据时可以更新估计值。

先验的影响以及如何选择合适的先验

从前面的例子可以看出，先验对分析的结果会有影响。一些贝叶斯分析的新手（以及一些诋毁该方法的人）会对如何选择先验感到茫然，因为他们不希望先验起到决定性作用，而是更希望数据本身替自己说话！有这样的想法很正常，不过我们得牢记，数据并不会真的“说话”，只有在模型中才会有意义，包括数学上的和脑海中的模型。面对同一主题下的同一份数据，不同人会有不同的看法，

这类例子在科学史上有许多，查看以下链接可以了解《纽约时报》最近一次实验的例 子：http://www.nytimes.com/interactive/2016/09/20/upshot/the-error-the-polling-world-rarely-talks-about.html?_r=0。

有些人青睐于使用没有信息量的先验（也称作均匀的、含糊的或者发散的先验），这类先验对分析过程的影响最小。本书将遵循 Gelman、McElreath 和 Kruschke 3 人的建议[①]，更倾向于使用带有较弱信息量的先验。在许多问题中，我们对参数可以取的值一般都会有些了解，比如，参数只能是正数，或者知道参数近似的取值范围，又或者是希望该值接近 0 或大于 / 小于某值。这种情况下，我们可以给模型加入一些微弱的先验信息而不必担心该先验会掩盖数据本身的信息。由于这类先验会让后验近似位于某一合理的边界内，因此也被称作正则化先验。

当然，使用带有较多信息量的强先验也是可行的。视具体的问题不同，有可能很容易或者很难找到这类先验，例如在我工作的领域（结构生物信息学），人们会尽可能地利用先验信息，通过贝叶斯或者非贝叶斯的方式来了解和预测蛋白质的结构。这样做是合理的，原因是我们在数十年间已经从上千次精心设计的实验中收集了数据，因而有大量可信的先验信息可供使用。如果你有可信的先验信息，完全没有理由不去使用。试想一下，如果一个汽车工程师每次设计新车的时候，他都要重新发明内燃机、轮子乃至整个汽车，这显然不是正确的方式。

现在我们知道了先验有许多种，不过这并不能缓解我们选择先验时的焦虑。或许，最好是没有先验，这样事情就简单了。不过，不论是否基于贝叶斯，模型都在某种程度上拥有先验，即使这里的先验并没有明确表示出来。事实上，许多频率统计学方面的结果可以看做是贝叶斯模型在一定条件下的特例，比如均匀先验。让我们再仔细看看前面那幅图，可以看到蓝色后验分布的峰值与频率学分析中 θ 的期望值是一致的：

$$\hat{\theta} = y/N$$

注意，这里 $\hat{\theta}$ 是点估计而不是后验分布（或者其他类型的分布）。由此看出，

① 该 3 人分别是《Bayesian Data Analysis》《Statistical Rethinking: A Bayesian Course with Examples in R and Stan》和《Doing Bayesian Data Analysis》的主要作者。——译者注

你没办法完全避免先验，不过如果你在分析中引入先验，得到的会是概率分布分布而不只是最可能的一个值。明确引入先验的另一个好处是，我们会得到更透明的模型，这意味着更容易评判、（广义上的）调试以及优化。构建模型是一个循序渐进的过程，有时候可能只需要几分钟，有时候则可能需要数年；有时候整个过程可能只有你自己，有时候则可能包含你不认识的人。而且，模型复现很重要，而模型中透明的假设能有助于复现。

在特定分析任务中，如果我们对某个先验或者似然不确定，可以自由使用多个先验或者似然进行尝试。模型构建过程中的一个环节就是质疑假设，而先验就是质疑的对象之一。不同的假设会得到不同的模型，根据数据和与问题相关的领域知识，我们可以对这些模型进行比较，本书第 6 章模型比较部分会深入讨论该内容。

由于先验是贝叶斯统计中的一个核心内容，在接下来遇到新的问题时我们还会反复讨论它，因此，如果你对前面的讨论内容感到有些疑惑，别太担心，要知道人们在这个问题上已经困惑了数十年并且相关的讨论一直在继续。

1.3.2　报告贝叶斯分析结果

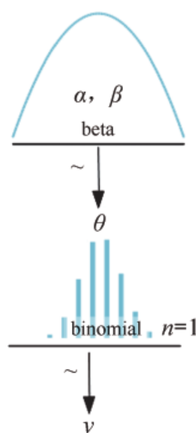
现在我们已经有了后验，相关的分析也就结束了。下面我们可能还需要对分析结果进行总结，将分析结果与别人分享，或者记录下来以备日后使用。

1.3.3　模型注释和可视化

根据受众不同，你可能在交流分析结果的同时还需要交流模型。以下是一种简单表示概率模型的常见方式：

- $\theta \sim Beta(\alpha, \beta)$
- $y \sim Bin(n=1, p=\theta)$

这是我们抛硬币例子中用到的模型。符号～表示左边随机变量的分布服从右边的分布形式，也就是说，这里 θ 服从于参数为 α 和 β 的 Beta 分布，而 y 服从于参数为 $n = 1$ 和 $p = \theta$ 的二项分布。该模型还可以用类似 Kruschke 书中的图表示成如下形式：



在第一层，根据先验生成了 θ，然后通过似然生成最下面的数据。图中的箭头表示变量之间的依赖关系，符号～表示变量的随机性。

本书中用到的类似 Kruschke 中的图都是由 Rasmus Bååth（http://www.sumsar.net/blog/2013/10/diy-kruschke-style-diagrams/）提供的模板生成的。

1.3.4 总结后验

贝叶斯分析的结果是后验分布，该分布包含了有关参数在给定数据和模型下的所有信息。如果可能的话，我们只需要将后验分布展示给观众即可。通常，一个不错的做法是：同时给出后验分布的均值（或者众数、中位数），这样能让我们了解该分布的中心，此外还可以给出一些描述该分布的衡量指标，如标准差，这样人们能对我们估计的离散程度和不确定性有一个大致的了解。拿标准差衡量类似正态分布的后验分布很合适，不过对于一些其他类型的分布（如偏态分布）却可能得出误导性结论，因此，我们还可以采用下面的方式衡量。

最大后验密度

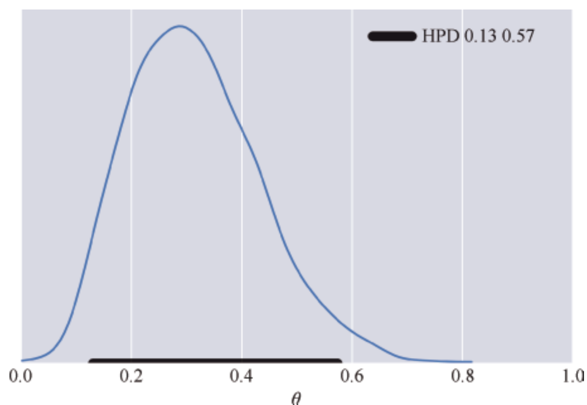
一个经常用来描述后验分布分散程度的概念是**最大后验密度**（Highest Posterior Density，HPD）区间。一个 HPD 区间是指包含一定比例概率密度的最小区间，最常见的比例是 95%HPD 或 98%HPD，通常还伴随着一个 50%HPD。如果我们说某个分析的 HPD 区间是 [2, 5]，其含义是指：根据我们的模型和数据，参数位于 2 ～ 5 的概率是 0.95。这是一个非常直观的解释，以至于人们经常会将频

率学中的置信区间与贝叶斯方法中的可信区间弄混淆。如果你对频率学的范式比较熟悉，请注意这两种区间的区别。贝叶斯学派的分析告诉我们的是参数取值的概率，这在频率学的框架中是不可能的，因为频率学中的参数是固定值，频率学中的置信区间只能包含或不包含参数的真实值。在继续深入之前，有一点需要注意：选择 95% 还是 50% 或者其他什么值作为 HPD 区间的概率密度比例并没有什么特殊的地方，这些不过是经常使用的值罢了。比如，我们完全可以选用比例为 91.37% 的 HPD 区间。如果你选的是 95%，这完全没问题，只是要记住这只是个默认值，究竟选择多大比例仍然需要具体问题具体分析。

对单峰分布计算 95%HPD 很简单，只需要计算出 2.5% 和 97.5% 处的值即可：

```
def naive_hpd(post):
    sns.kdeplot(post)
    HPD = np.percentile(post, [2.5, 97.5])
    plt.plot(HPD, [0, 0], label='HPD {:.2f} {:.2f}'.format(*HPD),
      linewidth=8, color='k')
    plt.legend(fontsize=16);
    plt.xlabel(r'$\theta$', fontsize=14)
    plt.gca().axes.get_yaxis().set_ticks([])

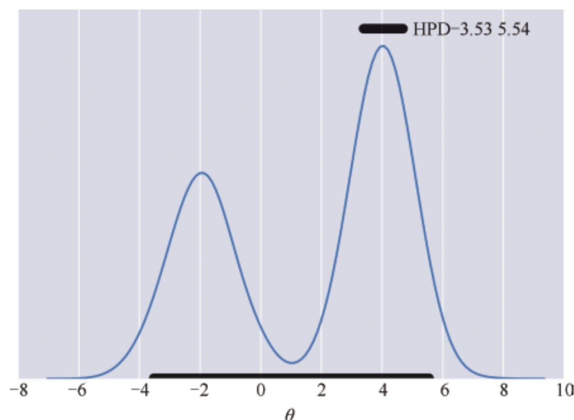

np.random.seed(1)
post = stats.beta.rvs(5, 11, size=1000)
naive_hpd(post)
plt.xlim(0, 1)
```

对于多峰分布而言，计算 HPD 要稍微复杂些。如果把对 HPD 的原始定义应用到混合高斯分布上，我们可以得到：

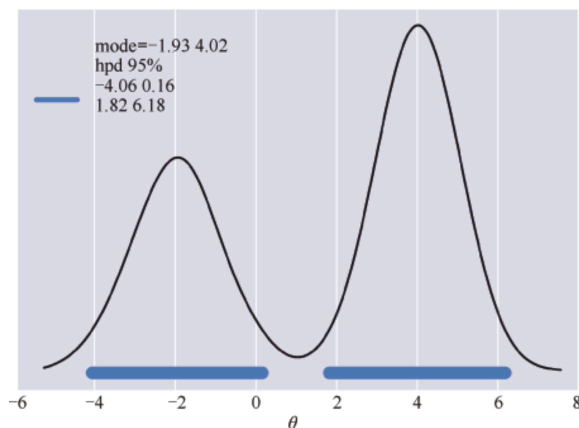
```
np.random.seed(1)
gauss_a = stats.norm.rvs(loc=4, scale=0.9, size=3000)
gauss_b = stats.norm.rvs(loc=-2, scale=1, size=2000)
mix_norm = np.concatenate((gauss_a, gauss_b))

naive_hpd(mix_norm)
plt.savefig('B04958_01_08.png', dpi=300, figsize=(5.5, 5.5))
```



从上图可以看出，通过原始 HPD 定义计算出的可信区间包含了一部分概率较低的区间，位于 [0,2]。为了正确计算出 HPD，这里我们使用了 `plot_post` 函数，你可以从本书附带的代码中下载对应的源码：

```
from plot_post import plot_post
plot_post(mix_norm, roundto=2, alpha=0.05)
plt.legend(loc=0, fontsize=16)
plt.xlabel(r"$\theta$", fontsize=14)
```



从上图可以看出，95%HPD 包含两个区间，同时 `plot_post` 函数也返回了两个众数。

1.4　后验预测检查

贝叶斯方法的一个优势是：一旦得到了后验分布，我们可以根据该后验生成未来的数据 y，即用来做预测。后验预测检查主要是对观测数据和预测数据进行比较从而发现这两个集合的不同之处，其目的是进行一致性检查。生成的数据和观测的数据应该看起来差不多，否则有可能是建模出现了问题或者输入数据到模型时出了问题，不过就算我们没有出错，两个集合仍然有可能出现不同。尝试去理解其中的偏差有助于我们改进模型，或者至少能知道我们模型的极限。即使我们并不知道如何去改进模型，但是理解模型捕捉到了问题或数据的哪些方面以及没能捕捉到哪些方面也是非常有用的信息。也许模型能够很好地捕捉到数据中的均值但却没法预测出罕见值，这可能是个问题，不过如果我们只关心均值，这个模型对我们而言也还是可用的。通常我们的目的不是去声称一个模型是错误的，我们更愿意遵循 George Box 的建议，即所有模型都是错的，但某些是有用的。我们只想知道模型的哪个部分是值得信任的，并测试该模型是否在特定方面符合我们的预期。不同学科对模型的信任程度显然是不同的，物理学中研究的系统是在高可控条件下依据高深理论运行的，因而模型可以看做是对现实的不错描述，而在一些其他学科如社会学和生物学中，研究的是错综复杂的孤立系统，因而模型对系统的认知较弱。尽管如此，不论你研究的是哪一门学科，都需要对模型进行检查，利用后验预测和本章学到的探索式数据分析中的方法去检查模型。

1.5　安装必要的 Python 库

本书用到的代码是用 Python 3.5 写的，建议使用 Python 3 的最新版本运行，尽管大多数代码也能在更老的 Python 版本（包括 Python 2.7）上运行，不过可能会需要做些微调。

本书建议使用 Anaconda 安装 Python 及相关库，Anaconda 是一个用于科学计算的软件分发，你可以从以下链接下载并了解更多：https://www.continuum.io/

downloads。在系统上装好 Anaconda 之后，就可以通过以下方式安装 Python 库了：

conda install NamePackage

我们会用到以下 Python 库：

- Ipython 5.0；
- NumPy 1.11.1；
- SciPy 0.18.1；
- Pandas 0.18.1；
- Matplotlib 1.5.3；
- Seaborn 0.7.1；
- PyMC3 3.0。

在命令行中执行以下命令即可安装最新稳定版的 PyMC3：

```
pip install pymc3
```

1.6 总结

我们的贝叶斯之旅首先围绕统计建模、概率论和贝叶斯定理做了一些简短讨论，然后用抛硬币的例子介绍了贝叶斯建模和数据分析，借用这个经典例子传达了贝叶斯统计中的一些最重要的思想，比如用概率分布构建模型并用概率分布来表示不确定性。此外我们尝试揭示了如何选择先验，并将其与数据分析中的一些其他问题置于同等地位（怎么选择似然，为什么要解决该问题等）。本章的最后讨论了如何解释和报告贝叶斯分析的结果。本章我们对贝叶斯分析的一些主要方面做了简要总结，后面还会重新回顾这些内容，从而充分理解和吸收，并为后面理解更高级的内容打下基础。下一章我们会重点关注一些构建和分析更复杂模型的技巧，此外，还会介绍 PyMC3 并将其用于实现和分析贝叶斯模型。

1.7 练习

我们尚不清楚大脑是如何运作的，是按照贝叶斯方式？还是类似贝叶斯的某种方式？又或许是进化过程中形成的某种启发式的方式？不管如何，我们至少知

道自己是通过数据、例子和练习来学习的，尽管你可能对此有不同的意见，不过我仍然强烈建议你完成以下练习。

（1）修改生成本章的第 3 个图的代码，在图中增加一条竖线用来表示观测到的正面朝上的比例（正面朝上的次数 / 抛硬币的次数），将该竖线的位置与每个子图中后验的众数进行比较。

（2）尝试用不同的先验参数（beta 值）和不同的实验数据（正面朝上的次数和实验次数）重新绘制本章的第 3 个图。

（3）阅读维基百科上有关 Cromwell 准则的内容：https://en.wikipedia.org/wiki/Cromwell%27s_rule。

（4）探索不同参数下高斯分布、二项分布和 beta 分布的图像，你可以为每个分布单独画一个图而不是全都画在一个网格中。



第 2 章
概率编程——PyMC3 编程指南

前面我们已经对贝叶斯统计有了初步了解，接下来将学习如何利用一些计算工具构建概率模型，还将学习概率编程。主要目的是利用代码描述模型并用其进行推断，当然这并非因为我们太懒了所以才没有采用数学的方法去推断，也不是因为我们是最棒的极客（做梦都在写代码……）。该决定主要基于这样一个原因：许多模型都没有一个封闭的解析后验，也就是说我们只能使用数值的方法计算后验。学习概率编程的另一个原因是：现代的贝叶斯统计主要是通过编程实现的，概率编程提供了一个有效的方式去构建复杂模型，它让我们更加关注模型设计、评估和解释，而不用过多地考虑数学或计算细节。

本章我们将学习贝叶斯统计中的数值计算方法以及如何使用 PyMC3。PyMC3 是概率编程中一个非常灵活的库，了解 PyMC3 有助于我们从更实用的角度学习更高级的贝叶斯概念。

本章涵盖以下主题：

- 概率编程；
- 推断引擎；
- PyMC3 指南；
- 重温抛硬币问题；
- 模型检查和诊断。

2.1 概率编程

贝叶斯统计的概念很简单，我们有一些固定的数据（固定的意思是指我们无法改变观测值），和一些感兴趣的参数，剩下要做的就是探索这些参数可能的取值，其中所有的不确定性都通过概率进行建模。在别的统计学范式中，未知量有

多种不同的表示方式，而在贝叶斯的框架中，所有未知量都是同等对待的，如果不知道某个变量，我们就给其赋予一个概率分布。因此，贝叶斯定理就是将先验概率分布 $p(\theta)$（在观测到数据之前我们对问题的理解）转化成后验分布 $p(\theta|D)$（观测到数据之后所得到的信息），换句话说，贝叶斯统计就是一种机器学习的过程。

尽管概念上很简单，纯粹的概率模型得到的表达式通常分析起来很棘手。许多年来，这都是一个问题，大概也是影响贝叶斯方法广泛应用的最大原因之一。随着计算时代的到来，数值化方法的发展使得计算几乎任意模型的后验成为了可能，这极大地促进了贝叶斯方法的应用。我们可以把这些数值化方法当作通用引擎，按照 PyMC3 的核心开发者之一 Thomas Wiecki 的说法，只需要按下按钮，推断部分就可以自动完成了。

自动化推断促进了概率编程语言的发展，从而使得模型构建和推断相分离。在概率编程语言的框架中，用户只需要寥寥数行代码描述概率模型，后面的推断过程就能自动完成了。概率编程使得人们能够更快速地构建复杂的概率模型并减少出错的可能，可以预见，这将给数据科学和其他学科带来极大的影响。

我认为，编程语言对科学计算的影响可以与 60 多年前 Fortran 语言的问世相对比。尽管如今 Fortran 语言风光不再，不过在当时 Fortran 语言被认为是相当革命性的。科学家们第一次从计算的细节中解放出来，开始用一种更自然的方式关注构建数值化的方法、模型和仿真系统。类似地，概率编程将处理概率和推断的过程对用户隐藏起来，从而使得用户更多的去关注模型构建和结果分析。

2.1.1　推断引擎

即便某些情况下不太可能从分析的角度得到后验，我们也有办法将后验计算出来，其中一些方法列举如下。

（1）非马尔科夫方法：

- 网格计算；
- 二次近似；

- 变分方法。

（2）马尔科夫方法：

- Metropolis-Hastings 算法；
- 汉密尔顿蒙特卡洛方法（Hamiltonian Monte Carlo）/ 不掉向采样（No U-Turn Sampler，NUTS）。

如今，贝叶斯分析主要是通过马尔科夫链蒙特卡洛（Markov Chain Monte Carlo，MCMC）方法进行，同时变分方法也越来越流行，特别是在一些较大的数据集上。学习贝叶斯分析并不需要完全掌握这些方法，不过从概念层面上对它们的工作原理有一定了解会很有帮助，比如调试模型。

非马尔科夫方法

我们首先讨论基于非马尔科夫方法的推断引擎。非马尔科夫方法在低维的问题上要比马尔科夫方法更快，对于某些问题，这类方法非常有用，而对另外一些问题，这类方法只能提供真实后验的粗略近似，不过没关系，它们可以为马尔科夫方法提供一个不错的初始点，后面我们会介绍马尔科夫的含义。

1. 网格计算

网格计算是一种暴力穷举的方法。即便你无法计算出整个后验，你也可以根据一些点计算出先验和似然。假设我们要计算某个单参数模型的后验，网格近似可以按照如下方式进行：

- 确定参数的一个合理区间（先验会给你点提示）；
- 在以上区间确定一些网格点（通常是等距离的）；
- 对于网格中的每个点计算先验和似然。

视情况，我们可能会对计算结果进行归一化（把每个点的计算结果除以所有点的计算结果之和）。

很容易看出，选的点越多（网格越密）近似的结果就越好。事实上，如果使用无限多的点，我们可以得到准确的后验。网格计算的方法不能很好地适用于多参数（又或者称多维度）的场景，随着参数的增加，采样空间相比后验空间会急剧增加，换言之，我们花费了大量时间计算后验值，但对于估计后验却几乎没有

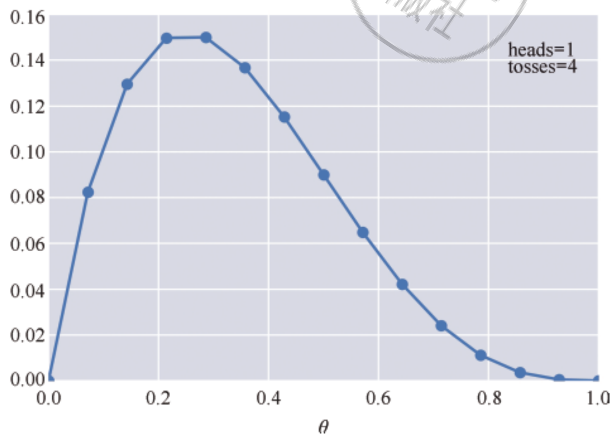
帮助，因而使得该方法对于大多数统计学和数据科学的问题都不太实用[①]。

下面的代码用网格计算的方法解决第一章中的抛硬币问题：

```
def posterior_grid(grid_points=100, heads=6, tosses=9):
    """
    A grid implementation for the coin-flip problem
    """
    grid = np.linspace(0, 1, grid_points)
    prior = np.repeat(5, grid_points)
    likelihood = stats.binom.pmf(heads, tosses, grid)
    unstd_posterior = likelihood * prior
    posterior = unstd_posterior / unstd_posterior.sum()
    return grid, posterior
```

假设 4 次抛硬币实验中只有一次观测到正面朝上，那么就有：

```
points = 15
h, n = 1, 4
grid, posterior = posterior_grid(points, h, n)
plt.plot(grid, posterior, 'o-', label='heads = {}\ntosses = {}'.format(h, n))
plt.xlabel(r'$\theta$')
plt.legend(loc=0)
```

2. 二次近似

二次近似，也称**拉普拉斯方法**或者正态近似，利用的是高斯分布来近似后验。该方法通常很有效，原因是后验分布众数附近的区域通常是服从正态分布

① 可阅读 https://zh.wikipedia.org/wiki/ 维数灾难，了解更多信息。——译者注

的，事实上很多情况下就是高斯分布。二次近似的计算过程分为两步。首先，找出后验分布的峰值，该值可以通过一些最优化算法得到（有许多开箱即用的算法用来求解函数的最大值或最小值），这样我们得到了用来近似的高斯分布的均值；然后估计函数在众数附近的曲率，根据该曲率可以得出近似高斯分布的标准差。在介绍完 PyMC3 之后，我们会讲解如何使用该方法。

3. 变分方法

现代的贝叶斯统计学大多都采用马尔科夫方法（下一节会介绍），不过该方法对某些问题求解很慢而且不能很好地并行计算。一种简单的做法是同时运行多个马尔科夫链，然后将结果合并，不过对大多数问题而言这并不是一个合适的解决方案，如何找到有效的并行计算方式是当前的一个研究热点。

对于较大的数据集或是某些计算量很大的似然而言，变分方法是一个更好的选择。此外，这类方法能快速得到后验的近似，为 MCMC 方法提供初始点。

变分方法的基本思想是用一个更简单的分布去近似后验，这听起来有点像拉普拉斯近似，不过深入细节你会发现二者有很大不同。变分方法的最大缺点是我们必须对每个模型设计一个特定的算法，因而变分方法并不是一个通用的推断引擎，而是与模型相关的。

当然，许多人都在尝试将变分方法自动化。最近提出的一个方法是**自动差分变分推断**（Automatic Differentiation Variational Inference，ADVI）（查看 http://arxiv.org/abs/1603.00788 了解更多）。从概念层面上讲，ADVI 是按下面步骤工作的。

（1）对参数进行变换从而使得它们位于整个实轴上。例如，假设一个参数限定为正数，对其求 log 后得到的值位于无界区间 $[-\infty,+\infty]$ 内。

（2）用高斯分布对无界参数近似。需要注意，转换后参数空间里的高斯分布在原来参数空间内是非高斯的，这也是与拉普拉斯近似方法的不同点。

（3）采用某种优化算法使得高斯近似的结果尽可能接近后验，该过程通过最大化**证据下界**（Evidence Lower Bound，ELBO）实现。如何衡量两个分布的相似性以及证据下界的具体含义都是一些数学细节。

ADVI 算法已经在 PyMC3 中实现了，本书后面会使用到。

马尔科夫方法

有许多相关的方法都称 MCMC 方法。对于网格计算近似方法而言，我们需要根据给定的点计算出似然和先验，并且近似出整个后验分布。MCMC 方法要比网格近似更好，这是因为其设计思想是将更多的时间用于高概率区域的计算，而在较低概率区间花费较少时间。事实上，MCMC 方法会根据相对概率访问参数空间中的不同区间。如果区间 A 的概率是区间 B 的两倍，那么我们从区间 A 采样的次数可能是从区间 B 采样次数的两倍。因此，即使我们无法从分析的角度得到整个后验，我们也可以采用 MCMC 的方法从中采样，并且采样数越多，效果越好。

要理解 MCMC 方法，我们先将其拆分成两个 MC 部分，即**蒙特卡洛**部分和**马尔科夫链**部分。

1. 蒙特卡洛

蒙特卡洛这部分可以用随机数的应用来解释。蒙特卡洛方法是一系列应用非常广泛的算法，其思想是通过随机采样来计算或模拟给定过程。蒙特卡洛是位于摩纳哥公国的一个非常有名的城市，蒙特卡洛方法的开发者之一是 Stanislaw Ulam。Stan 正是基于这一核心思想——尽管很多问题都难以求解甚至无法准确用公式表达，但我们可以通过采样或者模拟来有效地研究。据传，起因是为了计算单轮游戏中的概率，解决该问题的方法之一是组合分析法；另一种是 Stan 声称的，进行多次单轮游戏，最后计算其中有多少次是我们感兴趣的。这听起来似乎是显而易见的，或者至少是相当合理的，比如，你可能已经用重采样的方法来解决统计问题。不过这个实验是早在 70 年前进行的，当时，第一台计算机才刚开始研发。该方法首次应用于一个核物理问题。如今，个人计算机都已经足够强大，用蒙特卡洛方法可以解决许多有趣的问题，因此，该方法也被广泛应用到科学、工程、工业以及艺术等各个方面。

在使用蒙特卡洛方法计算数值的例子中，一个教科书上非常经典的是估计 π。实际使用中有更好的方法来计算 π，不过这个例子仍然具有教学意义。我们可以通过以下过程估计 π：

（1）在边长为 $2R$ 的正方形内随机撒 N 个点。

（2）在正方形内画一个半径为 R 的圆，计算在圆内的点的个数。

（3）得出 $\hat{\pi}$ 的估计值 $\frac{4 \times inside}{N}$。

需要注意：如果一个点满足以下条件，我们认为该点位于圆内：

$$\sqrt{(x^2 + y^2)} \leqslant R$$

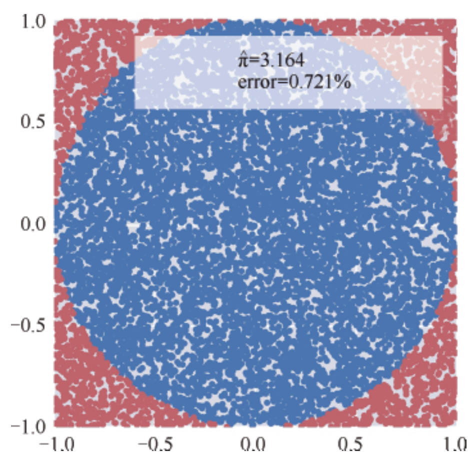
正方形的面积是 $(2R)^2$，圆的面积是 πR^2，因此二者的面积之比是 $4/\pi$，而圆和正方形的面积分别正比于圆内的点的个数和总的点数 N。我们可以通过几行简单的代码来模拟该蒙特卡洛过程计算 π 值，同时计算出估计值与实际值之间的相对误差：

```
N = 10000

x, y = np.random.uniform(-1, 1, size=(2, N))
inside = (x**2 + y**2)  <= 1
pi = inside.sum()*4/N
error = abs((pi - np.pi)/pi)* 100

outside = np.invert(inside)

plt.plot(x[inside], y[inside], 'b.')
plt.plot(x[outside], y[outside], 'r.')
plt.plot(0, 0, label='$\hat \pi$ = {:4.3f}\nerror = {:4.3f}%'.
format(pi, error), alpha=0)
plt.axis('square')
plt.legend(frameon=True, framealpha=0.9, fontsize=16);
```



上面的代码中，`outside` 是用来绘图的，在计算 $\frac{4 \times inside}{N}$ 的过程中没有用到。另外一点需要澄清的是，由于这里用的是单位圆，因此在判断一个点是否在圆内时没有计算平方根。

2. 马尔科夫链

马尔科夫链是一个数学对象，包含一系列状态以及状态之间的转移概率，如果每个状态转移到其他状态的概率只与当前状态相关，那么这个状态链就称为马尔科夫链。有这样一个马尔科夫链之后，我们可以任取一个初始点，然后根据状态转移概率进行随机游走。假设能够找到这样一个马尔科夫链，其状态转移概率正比于我们想要采样的分布（如贝叶斯分析中的后验分布），采样过程就变成了简单地在该状态链上移动的过程。那么，如何在不知道后验分布的情况下找到这样的状态链呢？有一个概念叫做**细节平衡条件**（Detailed Balance Condition），直观上讲，这个条件是说，我们需要采用一种可逆的方式移动（可逆过程是物理学中的一个常见的近似）。也就是说，从状态 i 转移到状态 j 的概率必须和状态 j 转移到状态 i 的概率相等。

总的来说就是，如果我们能够找到满足细节平衡条件的马尔科夫链，就可以保证从中采样得到的样本来自正确的分布。保证细节平衡的最流行的算法是 **Metropolis-Hasting 算法**。

3. Metropolis-Hasting 算法

为了更形象地理解这个算法，我们用下面这个例子来类比。假设我们想知道某个湖的水容量以及这个湖中最深的点，湖水很浑浊以至于没法通过肉眼来估计深度，而且这个湖相当大，网格近似法显然不是个好办法。为了找到一个采样策略，我们请来了两个好朋友小马和小萌。经过讨论之后想出了如下办法，我们需要一个船（当然，也可以是竹筏）和一个很长的棍子，这比声呐可便宜多了，而且我们已经把有限的钱都花在了船上。

（1）随机选一个点，然后将船开过去。

（2）用棍子测量湖的深度。

（3）将船移到另一个地点并重新测量。

（4）按如下方式比较两点的测量结果。

- 如果新的地点比旧的地点水位深，那么在笔记本上记录下新的测量值并重复过程（2）。
- 如果新的地点比旧的地点水位浅，那么我们有两个选择：接受或者拒绝。接受意味着记录下新的测量值并重复过程（2）；拒绝意味着重新回到上一个点，再次记录下上一个点的测量值。

如何决定接受还是拒绝新的测量值呢？这里的一个技巧便是使用 Metropolis-Hastings 准则，即接受新的测量值的概率正比于新旧两点的测量值之比。

按照以上过程迭代下去，我们不仅可以得到整个湖的水容量和最深的点，而且可以得到整个湖底的近似曲率。你也许已经猜到了，在这个类比中，湖底的曲率其实就是后验的分布，而最深的点就是后验的众数。根据小马的说法，迭代的次数越多，近似的效果越好。

事实上，理论保证了在这种情形下，如果我们能采样无数次，最终能得到完整的后验。幸运地是，实际上对于很多问题而言，我们只需要相对较少地采样就可以得到一个相当准确的近似。

现在让我们从更正式的角度来看看该算法。对于很多分布而言（如高斯分布），我们有相当高效的算法对其采样，但对于一些其他分布，情况就变了。Metropolis-Hastings 算法使得我们能够从任意分布中以概率 $p(x)$ 得到采样值，只要我们能算出某个与 $p(x)$ 成比例的值。这一点很有用，因为在类似贝叶斯统计的许多问题中，最难的部分是计算归一化因子，也就是贝叶斯定理中的分母。Metropolis-Hastings 算法的步骤如下。

（1）给参数 x_i 赋一个初始值，通常是随机初始化或者使用某些经验值。

（2）从某个简单的分布 $Q(x_{i+1} \mid x_i)$ 中选一个新的值 x_{i+1}，如高斯分布或者均匀分布。这一步可以看做是对状态 x_i 的扰动。

（3）根据 Metropolis-Hastings 准则计算接受一个新的参数值的概率：

$$p_a(x_{i+1} \mid x_i) = \min\left(1, \frac{p(x_{i+1})q(x_i \mid x_{i+1})}{p(x_i)q(x_{i+1} \mid x_i)}\right)$$

（4）从位于区间 [0,1] 内的均匀分布中随机选一个值，如果第（3）步中得到

的概率值比该值大，那么就接受新的值，否则仍保持原来的值。

（5）回到第（2）步重新迭代，直到我们有足够多的样本，稍后会解释什么叫足够多。

有几点需要注意。

- 如果选取的分布 $Q(x_{i+1} \mid x_i)$ 是对称的，那么可以得到 $p_{\alpha}(x_{i+1} \mid x_i) = \min\left(1, \frac{p(x_{i+1})}{p(x_i)}\right)$，这通常称为 **Metropolis 准则**。
- 步骤（3）和步骤（4）表明：我们总是会转移到一个比当前状态（或参数）概率更大的状态（或参数），对于概率更小的，则会以 x_{i+1} 与 x_i 之比的概率接受。该准则中的接受步骤使得采样过程相比网格近似方法更高效，同时保证了采样的准确性。
- 目标分布（贝叶斯统计中的后验分布）是通过记录下来的采样值来近似的。如果我们接受转移到新的状态 x_{i+1}，那么我们就记录该采样值 x_{i+1}。如果我们拒绝转移到 x_{i+1}，那么我们就记录 x_i。

最后，我们会得到一连串记录值，有时候也称**采样链**或者**迹**。如果一切都正常进行，那么这些采样值就是后验的近似。在采样链中出现次数最多的值就是对应后验中最可能的值。该过程的一个优点是：后验分析很简单，我们把对后验求积分的过程转化成了对采样链所构成的向量求和的过程。

下面的代码展示了 Metropolis 算法的一个基本实现。这段代码并不是为了解决什么实际问题，只是在这里用来演示，如果我们知道怎么计算给定点的函数值，我们就可能得到该函数的采样。需要注意下面的代码中不包含贝叶斯相关的部分，既没有先验也没有数据。要知道，MCMC 是一类能够用于解决很多问题的通用方法。例如，在一个（非贝叶斯的）分子模型中，我们可能需要一个函数来计算在某个状态 x 下系统的能量而不是简单地调用 `func.pdf(x)` 函数。

`metropolis` 函数的第一个参数是一个 SciPy 的分布，假设我们不知道如何从中直接采样。

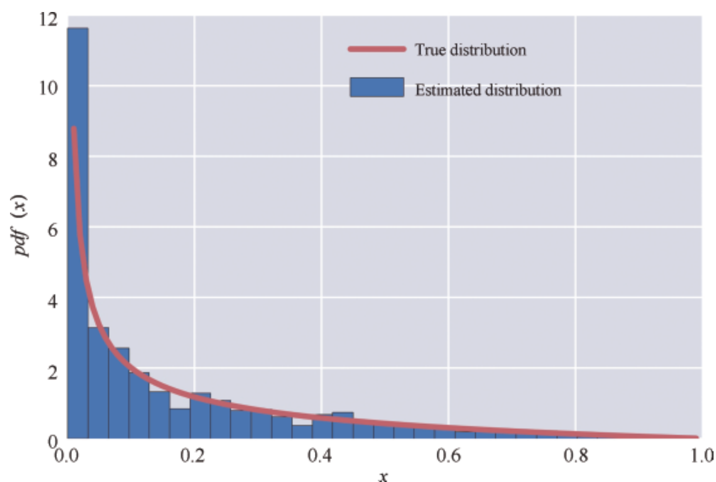
```
def metropolis(func, steps=10000):
    """A very simple Metropolis implementation"""
    samples = np.zeros(steps)
```

```
    old_x = func.mean()
    old_prob = func.pdf(old_x)

    for i in range(steps):
        new_x = old_x + np.random.normal(0, 0.5)
        new_prob = func.pdf(new_x)
        acceptance = new_prob/old_prob
        if acceptance >= np.random.random():
            samples[i] = new_x
            old_x = new_x
            old_prob = new_prob
        else:
            samples[i] = old_x
    return samples
```

下面的例子中，我们把 `func` 定义成一个 beta 函数，因为 beta 函数可以通过改变参数调整分布的形状。我们把 `metropolis` 函数的结果用直方图画出来，同时用红色的线表示真实的分布。

```
func = stats.beta(0.4, 2)
samples = metropolis(func=func)
x = np.linspace(0.01, .99, 100)
y = func.pdf(x)
plt.xlim(0, 1)
plt.plot(x, y, 'r-', lw=3, label='True distribution')
plt.hist(samples, bins=30, normed=True, label='Estimated distribution')
plt.xlabel('$x$', fontsize=14)
plt.ylabel('$pdf(x)$', fontsize=14)
plt.legend(fontsize=14)
```



现在你应该从概念上掌握了 Metropolis-Hastings 算法。也许你需要回过头去重新阅读前面几页才能完全消化。此外，我还强烈建议阅读 PyMC3 核心作者之一写的博文 http://twiecki.github.io/blog/2015/11/10/mcmc-sampling/[①]。他用一个简单的例子实现了 `metropolis` 方法，并将其用于求解后验分布，文中用非常好看的图展示了采样的过程，同时简单讨论了最初选取的步长是如何影响结果的。

4. 汉密尔顿蒙特卡洛方法 / 不掉向采样

MCMC 方法，包括 Metropolis-Hastings，都在理论上保证如果采样次数足够多，最终会得到后验分布的准确近似。不过，实际中想要采样足够多次可能需要相当长的时间，因此，人们提出了一些 Metropolis-Hastings 算法的替代方案。这些替代方案，包括 Metropolis-Hastings 算法本身，最初都是用来解决统计力学中的问题。统计力学是物理学的一个分支，主要研究原子和分子系统的特性。汉密尔顿蒙特卡洛方法，又称混合蒙特卡洛（Hybrid Monte Carlo，HMC），是这类改进方案之一。简单来说，汉密尔顿这个词描述的是物理系统的总能量，而另外一个名称中的“混合”是指将 Metropolis-Hastings 算法与分子力学（分子系统中广泛应用的一种仿真技巧）相结合。HMC 方法本质上和 Metropolis-Hastings 是一样的，改进的地方在于：原来是随机放置小船，现在有了一个更聪明的办法，将小船沿着湖底方向放置。为什么这个做法更聪明？因为这样做避免了 Metropolis-Hastings 算法的主要问题之一：探索得太慢而且采样结果自相关（因为大多数采样结果都被拒绝了）。

那么，如何才能不必深入其数学细节而理解汉密尔顿蒙特卡洛方法呢？假设我们还是在湖面上坐着船，为了决定下一步将要去哪，我们从当前位置往湖底扔了一个球，受“球状奶牛”的启发[②]，我们假设球面是理想的，没有摩擦，因而不会被泥巴和水减速。扔下球之后，让它滚一小会儿，然后把船划到球所在的位置。现在利用 Metropolis-Hastings 算法中提到的 Metropolis 准则来选择接受或者拒绝，重复整个过程一定次数。改进后的过程有更高的概率接受新的位置，即使

① 该文章 notebook 形式的翻译见 https://github.com/findmyway/Bayesian-Analysis-with-Python/blob/master/MCMC-sampling-for-dummies.ipynb。——译者注

② “球状奶牛”起源于一个物理学家的笑话，具体含义可自行搜索维基百科。——译者注

它们的位置相比前一位置距离较远。

现在跳出我们的思维实验，回到现实中来。基于汉密尔顿的方法需要计算函数的梯度。梯度是在多个维度上导数的推广。我们可以用梯度信息来模拟球在曲面上移动的过程。因此，我们面临一个权衡：HMC 计算过程要比 Metropolis-Hastings 更复杂，但是被接受概率更高。对于一些复杂的问题，HMC 方法更合适一些。HMC 方法的另一个缺点是：想要得到很好的采样需要指定一些参数。如果手动指定，需要反复尝试，这要求使用者有一定的经验。幸运地是，PyMC3 中有一个相对较新的不掉向采样算法，该方法被证实可以有效提升 HMC 方法的采样效率，同时不必手动调整参数。

5. 其他 MCMC 方法

MCMC 的方法很多，而且人们还在不断提出新的方法。如果你认为你能提升采样效率，会有很多人对你的想法感兴趣。讨论所有这类方法及其优缺点显然超出了本书的范围。不过，有些方法仍值得一提，因为你可能经常会听到人们讨论它们，所以最好是了解下他们都在讨论些什么。

在分子系统模拟中广泛应用的另外一种采样方法是**副本交换**（Replica Exchange），也称**并行退火**（Parallel Tempering）或者 **Metropolis Coupled MCMC**（MC3；好多 MC……）。该方法的基本思想是并行模拟多个不同的副本，每个副本都按照 Metropolis-Hastings 算法执行。多个副本之间的唯一不同是一个叫做温度的参数（又受到物理学的启发！），该参数用来控制接受低概率采样点的可能性。每隔一段时间，该方法都尝试在多个副本之间进行切换，切换过程同时遵循 Metropolis-Hastings 准则来接受或拒绝，只不过现在考虑的是不同副本之间的温度。切换过程可以在状态链上进行随机切换，不过，通常更倾向于在相邻副本之间切换，也就是说，具有相似温度的副本会有更高的接受概率。该方法的直观理解是：如果我们提高温度，接受新位置的概率也会提升，反之则会降低。温度更高的副本可以更自由地探索系统，因为这些副本的表面会变得相当平坦从而更容易探索。对于温度无限高的副本，所有状态都是等概率的。副本之间的切换避免了较低温度的副本陷在局部最低点，因而该方法很适合探索有多个最低点的系统。

2.2　PyMC3 介绍

PyMC3 是一个用于概率编程的 Python 库，当前最新的版本号是 2016 年 10 月 4 号发布的 3.0.rc2。PyMC3 提供了一套非常简洁直观的语法，非常接近统计学中描述概率模型的语法，可读性很高。PyMC3 是用 Python 写的，其中的核心计算部分基于 NumPy 和 Theano。Theano 是一个用于深度学习的 Python 库，可以高效地定义、优化和求解多维数组的数学表达式。PyMC3 使用 Theano 的主要原因是某些采样算法（如 NUTS）需要计算梯度，而 Theano 可以很方便地进行自动求导。而且，Theano 将 Python 代码转化成了 C 代码，因而 PyMC3 的速度相当快。关于 Theano 就需要了解这些，如果你想深入学习更多可以阅读 Theano 官网上的教程 http://deeplearning.net/software/theano/tutorial/index.html#tutorial。

2.2.1　用计算的方法解决抛硬币问题

让我们重新回顾下抛硬币问题，这次我们使用 PyMC3。首先我们需要获取数据，这里我们使用手动构造的数据。由于数据是我们自己生成，所以知道真实的参数 θ，以下代码中用 `theta_real` 变量表示。显然，在真实数据中，我们并不知道参数的真实值，而是要将其估计出来。

```
np.random.seed(123)
n_experiments = 4
theta_real = 0.35
data = stats.bernoulli.rvs(p=theta_real, size=n_experiments)
print(data)
array([1, 0, 0, 0])
```

模型描述

现在有了数据，需要再指定模型。回想一下，模型可以通过指定似然和先验的概率分布完成。对于似然，我们可以用参数分别为 $n=1$ 和 $p=\theta$ 的二项分布来描述，对于先验，我们可以用参数为 $\alpha=\beta=1$ 的 beta 分布描述。这个 beta 分布与 [0,1] 区间内的均匀分布是一样的。我们可以用数学表达式描述如下：

$$\theta \sim Beta(\alpha, \beta)$$
$$y \sim Bin(n=1, p=\theta)$$

这个统计模型与 PyMC3 的语法几乎一一对应。第 1 行代码先构建了一个模型的容器，PyMC3 使用 with 语法将所有位于该语法块内的代码都指向同一个模型，你可以把它看作是简化模型描述的“语法糖”，这里将模型命名为 our_first_model。第 2 行代码指定了先验，可以看到，语法与数学表示很接近。我们把随机变量命名为 θ，需要注意的是，这里变量名与 Beta 函数的第 1 个参数名一样；保持相同的名字是个好习惯，这样能避免混淆。然后，我们通过变量名从后验采样中提取信息。这里变量 θ 是一个随机变量，我们可以将该变量看做是从某个分布（在这里是 beta 分布）中生成数值的方法而不是某个具体的值。第 3 行代码用跟先验相同的语法描述了似然，唯一不同的是我们用 observed 变量传递了观测到的数据，这样就告诉了 PyMC3 我们的似然。其中，data 可以是一个 Python 列表或者 Numpy 数组或者 Pandas 的 DataFrame。这样我们就完成了模型的描述。

```
with pm.Model() as our_first_model:
    theta = pm.Beta('theta', alpha=1, beta=1)
    y = pm.Bernoulli('y', p=theta, observed=data)
```

按下推断按钮

对于抛硬币这个问题，后验分布既可以从分析的角度计算出来，也可以通过 PyMC3 用几行代码从后验的采样中得到。代码中的第 1 行，调用了 find_MAP 函数，该函数调用 SciPy 中常用的优化函数尝试返回最大后验（Maximum a Posteriori，MAP）。调用 find_MAP 是可选的，有时候其返回值能够为采样方法提供一个不错的初始点，不过有时候却并没有多大用，因此大多数时候会避免使用它。然后，下一行定义了采样方法。这里用的是 Metropolis-Hastings 算法，函数名取了简写 Metropolis。PyMC3 可以让我们将不同的采样器赋给不同的随机变量；眼下我们的模型只有一个参数，不过后面我们会有多个参数。我们也可以省略该行，PyMC3 会根据不同参数的特性自动地赋予一个采样器，例如，NUTS 算法只对连续变量有用，因而不能用于离散的变量，Metropolis 算法能够处理离散的变量，而另外一些类型的变量有专门的采样方法。总的来说，我们可以让 PyMC3 为我们选一个采样方法。最后一行是执行推断，其中第 1 个参数是采样次数，第 2 个和第 3 个参数分别是采样方法和初始点，可以看到这两个参数是可选的。

```
    start = pm.find_MAP()
```

```
    step = pm.Metropolis()
    trace = pm.sample(1000, step=step, start=start)
```

这样，只需要几行代码我们就完成了整个模型的描述和推断。感谢 PyMC3 的开发者们为我们提供了这么棒的库。

诊断采样过程

现在我们根据有限数量的样本对后验做出了近似，接下来要做的第一件事就是检查我们的近似是否合理。我们可以做一些测试，有些是可视化的，有些是定量的。这些测试会尝试从样本中发现问题，但并不能证明我们得到的分布是正确的，它们只能提供证据证明样本看起来是合理的。如果我们通过样本发现了问题，解决办法有如下几种。

- 增加样本次数。
- 从样本链（迹）的前面部分去掉一定数量的样本，称为**老化**（Burn-in）。在实践中，MCMC 方法通常需要经过一段时间的采样之后，才得到真正的目标分布。老化在无限多次的采样中并不是必须的，因为这部分并没有包含在马尔科夫理论中。事实上，去掉前面部分的样本只不过是在有限次采样中提升结果的一个小技巧。需要注意，不要被数学对象和数学对象的近似弄糊涂了，球体、高斯分布以及马尔科夫链等数学对象只存在于柏拉图式的想象世界中，并不存在于不完美但却真实的世界中。
- 重新参数化你的模型，也就是说换一种不同但却等价的方式描述模型。
- 转换数据。这么做有可能得到更高效的采样。转换数据的时候需要注意对结果在转换后的空间内进行解释，或者再重新转换回去，然后再解释结果。

本书剩余部分将会详细讲解这些方案。

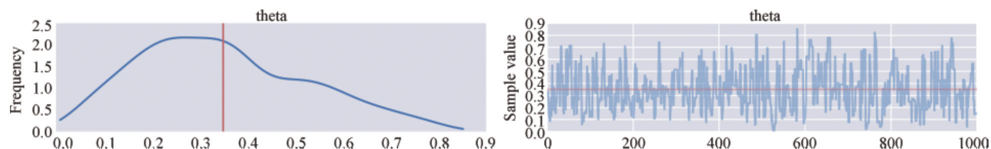
收敛性

通常，我们要做的第一件事就是查看结果长什么样，`traceplot` 函数非常适合该任务：

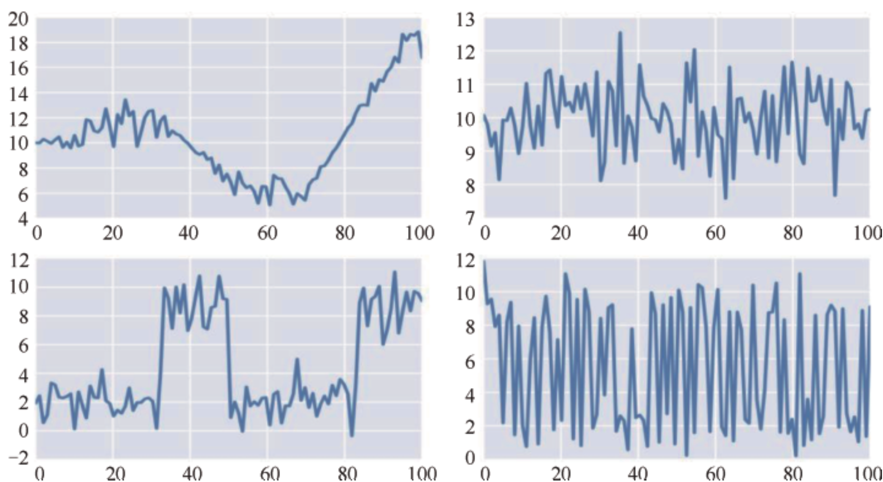
```
burnin = 100
chain = trace[burnin:]
```

```
pm.traceplot(chain, lines={'theta':theta_real});
```

对于未观测到的变量，我们得到了两幅图。左图是一个**核密度估计**（Kernel Density Estimation，KDE）图，可以看做是平滑之后的直方图。右图描绘的是每一步采样过程中得到的采样值。注意图中红色的线表示变量 `theta_real` 的值。



在得到这些图之后，我们需要观察什么呢？首先，KDE 图看起来应该是光滑的曲线。通常，随着数据的增加，根据中心极限定理[①]，参数的分布会趋近于高斯分布。当然，这并不一定是正确的。右侧的图看起来应该像白噪声，也就是说有很好的**混合度**（**mixing**），我们看不到任何可以识别的模式，也看不到向上或者向下的曲线，相反，我们希望看到曲线在某个值附近震荡。对于多峰分布或者离散分布，我们希望曲线不要在某个值或区域停留过多时间，我们希望看到采样值在多个区间自由移动。此外，我们希望迹表现出稳定的相似性，也就是说，前 10% 看起来跟后 50% 或者 10% 差不多。再次强调，我们不希望看到规律的模式，相反我们期望看到的是噪声。下图展示了一些迹呈现较好混合度（右侧）与较差混合度（左侧）的对比。



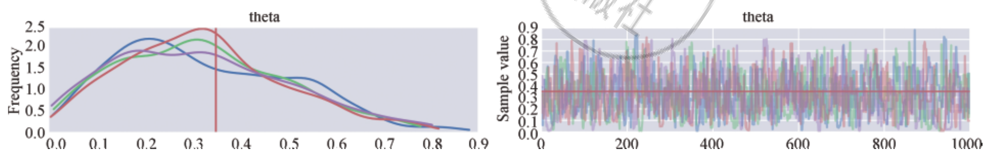
① 此处已根据原书的勘误更正。——译者注

如果迹的前面部分跟其他部分看起来不太一样，那就意味着需要进行老化处理，如果其他部分没有呈现稳定的相似性或者可以看到某种模式，那这意味着需要更多的采样，或者需要更换采样方法或者参数化方法。对于一些复杂的模型，我们可能需要结合使用前面所有的策略。

PyMC3 可以实现并行地运行一个模型多次，因而对同一个参数可以得到多条并行的迹。这可以通过在采样函数中指定 `njobs` 参数实现。此时使用 `traceplot` 函数，便可在同一幅图中得到同一个参数的所有迹。由于每组迹都是相互独立的，所有的迹看起来都应该差不多。除了检查收敛性之外，这些并行的迹也可以用于推断，我们可以将这些并行的迹组合起来提升采样大小而不是扔掉多余的迹：

```
with our_first_model:
    step = pm.Metropolis()
    multi_trace = pm.sample(1000, step=step, njobs=4)

burnin = 0
multi_chain = multi_trace[burnin:]
pm.traceplot(multi_chain, lines={'theta':theta_real});
```

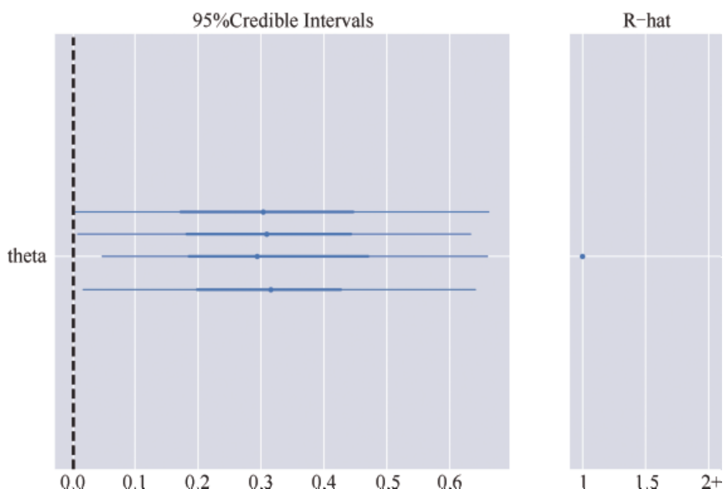



一种定量地检测收敛性的方法是 **Gelman-Rubin** 检验。该检验的思想是比较不同迹之间的差异和迹内部的差异，因此，需要多组迹来进行该检验。理想状态下，我们希望得到 $\hat{R}=1$。根据经验，我们认为如果得到的值低于 1.1，那么可以认为是收敛的了，更高的值则意味着没有收敛：

```
pm.gelman_rubin(multi_chain)
{'theta': 1.0074579751170656, 'theta_logodds': 1.009770031607315}
```

我们还可以用 `forestplot` 函数将 $\hat{R}$ 和每个参数的均值、50%HPD 和 95%HPD 可视化地表示出来：

```
pm.forestplot(multi_chain, varnames=['theta']);
```



函数 summary 提供了对后验的文字描述，它可以提供后验的均值、标准差和 HPD 区间：

```
pm.summary(multi_chain)
theta:
  Mean                SD              MC Error          95% HPD interval
  -------------------------------------------------------------------
  0.339               0.173           0.006              [0.037, 0.659]
Posterior quantiles:
  2.5            25                                 50
75          97.5
   |--------------|==============|==============|--------------|
  0.063        0.206                          0.318
0.455        0.709
```

此外，df_summary 函数会返回类似的结果，不过类型是 Pandas 中的 DataFrame：

```
pm.df_summary(multi_chain)
```

	mean	sd	mc_error	hpd_2.5	hpd_97.5
theta	0.33883	0.17305	0.00592	0.03681	0.65916

其中，返回值之一是 mc_error，这是对采样引入误差的估计值，该值考虑的是所有的采样值并非真的彼此独立。mc_error 是迹中不同块的均值的标准差，

每一块是迹中的一部分：

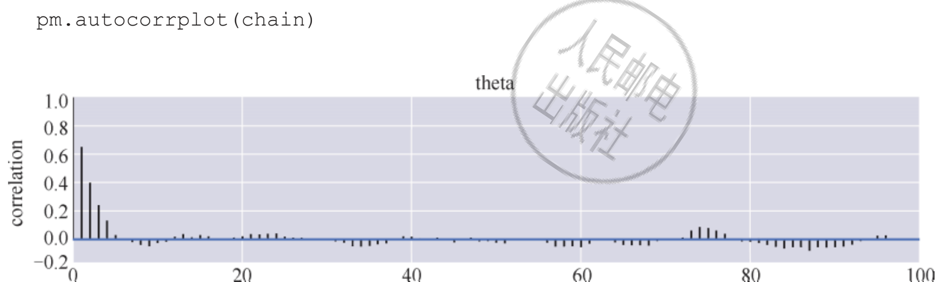
$$MC_{\text{error}} = \frac{\sigma(x)}{\sqrt{n}}$$

该误差值显然低于我们结果的准确度。由于采样方法是随机的，每次重跑的时候，summary 或者 df_summary 返回的值都会不同，不过没关系，mc_error 的值应该是相似的，如果返回的值有很大不同，则说明我们可能需要更多的样本。

自相关

最理想的采样应该不会是自相关的，也就是说，某一点的值应该与其他点的值是相互独立的。在实际中，从 MCMC 方法（特别是 Metropolis-Hastings）中得到的采样值是自相关的。由于参数之间的相互依赖关系，可能模型会导致更多的自相关采样。PyMC3 有一个很方便的函数用来描述自相关。

```
pm.autocorrplot(chain)
```



该图显示了采样值与相邻连续点（最多 100 个）之间的平均相关性。理想状态下，我们不会看到自相关性，实际中我们希望看到自相关性降低到较低水平。参数越自相关，要达到指定精度的采样次数就需要越多，也就是说，自相关性不利于降低采样次数。

有效采样大小

一个有自相关性的采样要比没有自相关性的采样所包含的信息量更少，因此，给定采样大小和采样的自相关性之后，我们可以尝试估计出该采样的大小为多少时，该采样没有自相关性而且包含的信息量不变，该值称为有效采样大小。理想情况下，两个值是一模一样的；二者越接近，我们的采样效率越高。有效采样大小可以作为我们的一个参考，如果我们想要估计出一个分布的均值，我们需

要的最小采样数至少为 100；如果想要估计出依赖于尾部分布的量，比如可信区间的边界，那么我们可能需要 1000 到 10000 次采样。

```
pm.effective_n(multi_chain)['theta']
667
```

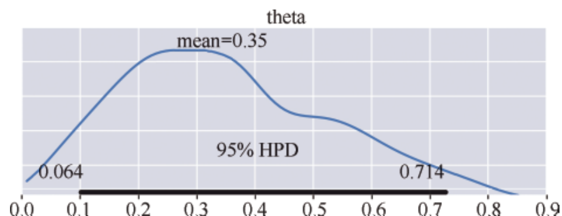
显然，提高采样效率的一个方法是换一个更好的采样方法；另一个办法是转换数据或者对模型重新设计参数，此外，还有一个常用的办法是对采样链压缩。所谓压缩其实就是每隔 k 个观测值取一个，在 Python 中我们称为切片。压缩会降低自相关性，但代价是同时降低了样本量。因此，实际使用中通常更倾向于增加样本量而不是切片。不过有时候，压缩会很有用，比如降低存储空间。如果仍不能避免高自相关性，我们就只能算出更长的采样链，模型中的参数很多的话，存储量会是个问题。而且，我们可能还会对后验做一些计算量很大的后处理，此时在自相关性尽可能小的前提下，采样数量的大小就显得尤为重要。

目前为止，所有的诊断测试都是经验性而非绝对的。实际使用中，我们会先运行一些测试，如果看起来没什么问题，我们就继续往下分析。如果发现了一些问题，就需要回过头解决它们，这也是建模过程的一部分。要知道，进行收敛性检查并非贝叶斯理论的一部分，由于我们是用数值的方式在计算后验，因而这只是贝叶斯实践过程中的一部分。

2.3 总结后验

我们已经知道，贝叶斯分析的结果是后验分布，其包含了在已有数据和模型下，参数的所有信息。我们可以使用 PyMC3 中的 `plot_posterior` 函数对后验分布进行可视化总结，这个函数的核心参数是一个 PyMC3 的迹和或者一个 NumPy 的数组，默认情况下，该函数会画出参数的直方图以及分布的均值，此外图像的底端还有一个黑色的粗线用来表示 95%HPD 区间。可以通过设置 `alpha_level` 参数来改变 HPD 区间。我们将这类图称为 Kruschke 图，这是因为 John K. Kruschke 第一次在他的《Doing Bayesian Data Analysis》一书中引入了这种图。

```
pm.plot_posterior(chain, kde_plot=True)
```



2.3.1　基于后验的决策

有时候，仅仅描述后验还不够，我们还需要根据推断结果做决策，即将连续的估计值收敛到一个二值化结果上：是或不是、受污染了还是安全的等等。回到抛硬币问题上，我们需要回答硬币是不是公平的。一枚公平的硬币是指 θ 的值为 0.5，严格来说，出现这种情况的概率是 0，因而，实际中我们会对定义稍稍放松，假如一枚硬币的 θ 值在 0.5 左右，我们就认为这枚硬币是公平的。这里“左右”的具体含义依赖于具体的问题，并没有一个满足所有问题的普适准则。因此决策也是偏主观的，我们的任务就是根据我们的目标做出最可能的决策。

直观上，一个明智的做法是将 HPD 区间与我们感兴趣的值进行比较，在我们的例子中，该值是 0.5。前面的图中可以看出 HPD 的范围是 0.06 ～ 0.71，包含 0.5 这个值，不过根据后验分布来看，硬币似乎倾向于反面朝上，我们无法就此裁定一个硬币是公平的。或许，我们需要收集更多的数据来降低数据的分散程度，从而得到一个更确定的决策；又或者是因为我们漏掉了某些关键信息，以至我们没能找到更完备的先验。

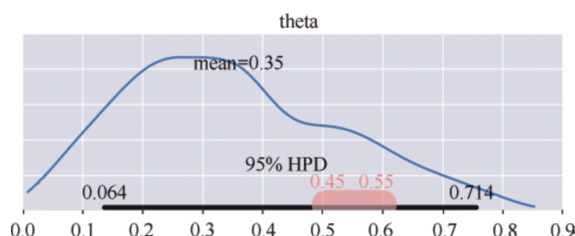
ROPE

基于后验做决策的一种方案是**实用等价区间**（Region Of Practical Equivalence，ROPE），其实就是在感兴趣值附近的一个区间，例如我们可以说 [0.45,0.55] 是 0.5 的一个实用性等价区间。同样，ROPE 是根据实际情况决定的。接下来我们可以将 ROPE 与 HPD 对比，结果至少有以下 3 种情况。

- ROPE 与 HPD 区间没有重叠，因此我们可以说硬币是不公平的。
- ROPE 包含整个 HPD 区间，我们可以认为硬币是公平的。
- ROPE 与 HPD 区间部分重叠，此时我们不能判断硬币是否公平。

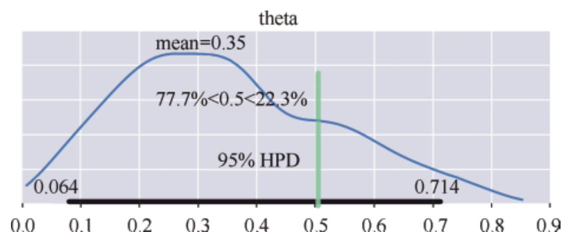
当然，如果选择区间 [0,1] 作为 ROPE，那么不管结果怎样我们都会说这枚硬币是公平的，不过恐怕没人会同意我们对 ROPE 的定义。plot_posterior 函数可以用来画 ROPE。从图中可以看到，ROPE 是一段较宽的半透明的红色线段，同时上面有两个数值表示 ROPE 的两个端点。

```
pm.plot_posterior(chain, kde_plot=True, rope=[0.45, 0.55])
```



我们还可以给 plot_posterior 传递一个参考值，例如 0.5，用来和后验进行对比。从图中可以看出我们会得到一个绿色的垂直线以及大于该值和小于该值的后验比例。

```
pm.plot_posterior(chain, kde_plot=True, ref_val=0.5)
```



关于如何使用 ROPE 的更多细节，你可以阅读 John Kruschke 写的《Doing Bayesian Data Analysis》一书的第 12 章。这一章还讨论了在贝叶斯框架下如何做假设检验，以及一些（贝叶斯或者非贝叶斯的）假设检验方面的警告。

损失函数

如果你觉得 ROPE 准则有些杂乱，想要更正式一些的形式，那么损失函数就是你想要的。做好决策很重要的一点是，参数的估计值要有很高的精度，同时还要考虑到预测出错的代价。对于收益 / 损失的权衡可以从数学形式上用代价函数来描述，有时候也称为损失函数。损失函数刻画的是当 Y（硬币是不公平的）成立时预测 X（硬币是公平的）成立时的代价。在许多问题中，决策的代价函数是

不对称的，例如，在决定 5 岁以下的儿童是否应该接种某种疫苗这件事上，决定接种或者不接种可能造成完全不同的影响，一旦做出错误的决策，可能会导致上千人死亡，而假如能决定接种某种非常安全同时又相对便宜的疫苗，则可能避免一场健康危机。人们对如何在有限信息下做出决策的问题已经研究许多年了，该学科被称为**决策理论**。

2.4　总结

本章，我们学习了概率编程，同时体会到了推断引擎的强大力量。我们从概念上讨论了 MCMC 方法的核心思想及其在现代贝叶斯数据分析中的地位。此外，我们第一次见证了 PyMC3 的强大和易用性。我们还重新回顾了前一章中的抛硬币问题，这次，我们使用 PyMC3 重新定义并解决了这个问题，同时还做了建模过程中非常重要的模型检查和模型诊断。

下一章会继续学习贝叶斯分析的一些技巧，我们将学习如何处理多个参数的模型，以及如何协调多个参数之间的层次关系。

2.5　深入阅读

- PyMC3 的文档，一定要记得查看例子部分：https://pymc-devs.github.io/pymc3/。
- 《贝叶斯方法——概率编程与贝叶斯推断》（注：该书已出版），这本书最初是用 PyMC2 写的，目前已经转成 PyMC3 了：https://github.com/quantopian/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers。
- 《While My MCMC Gently Samples》，PyMC3 核心开发者之一，Thomas Wiecki 的博客。
- 《Statistical Rethinking》，Richard McElreath 写的一本关于贝叶斯分析的入门书。书中的例子是用 R/Stan 写的，我将这本书中的例子转成了 Python/PyMC3，相关代码可以查看 https://github.com/aloctavodia/Statistical-Rethinking-with-Python-and-PyMC3。

- 《Doing Bayesian Data Analysis》，John K. Kruschke 写的另一本关于贝叶斯分析的入门书，也有跟前面一本书类似的问题，该书第 1 版中的大部分例子也都转成了 Python/PyMC3，相关代码可以查看 https://github.com/aloctavodia/Doing_bayesian_data_analysis。[①]

2.6 练习

（1）用其他先验尝试网格算法。例如，尝试将代码改成 `prior = (grid <= 0.5).astype(int)` 或者 `prior = abs(grid - 0.5)`，或者你可以将其定义成你所想的任意形式。换一些其他数据重复实验，例如增加总的样本数或者将样本数根据你看到的正面朝上的次数适当调整。

（2）在我们估计 p 值的代码中，将 N 固定，重复多次实验。注意由于我们使用了随机数，每次的结果都会不一样，不过检查一下可以发现误差应该差不多。尝试把 N 调大然后再重跑，你能猜出 N 与误差之间的关系吗？你可能需要修改下代码，将 N 作为一个参数传给计算误差的函数，这样方便估计 N 与误差之间的关系。对于相同的 N 值，可以重复跑多次后计算误差的平均值和这些误差的标准差，然后使用 matplotlib 中的 `errorbar()` 函数将它们画出来。可以尝试一些类似 100、1000、10000 的数作为 N 的值（每次提高一个数量级）。

（3）修改传给 `metropolis` 函数的参数。尝试使用第 1 章中用到的先验。将这段代码与网格计算进行比较，看一下哪部分需要修改后才能用于解决贝叶斯推断问题。

（4）将前一题的答案与下面链接中 Thomas Wiechi 的代码进行比较：http://twiecki.github.io/blog/2015/11/10/mcmc-sampling/。

（5）修改 beta 先验分布的参数以匹配前 1 章中的分布，试比较 2 者的结果。将 beta 分布替换成区间为 [0,1] 的均匀分布，其结果是否与 $beta(a=1,b=1)$ 相等？采样速度相比是更快？还是更慢？或者是相等？如果换成更大的区间，比如 [-1,2] 呢？模型是否能正常运行？你得到的错误是什么意思？如果你不调用 `find_MAP()` 函数的话呢？（记得将 `sample` 函数的第 1 个参数也去掉，如果你是在

① 第 2 版也有人转成了 Python，https://github.com/JWarmenhoven/DBDA-python。——译者注

Jupyter/IPython 记事本中运行代码，尤其需要注意这点。）

（6）修改退化的数量。尝试将退化的数量修改为 0 或者 500，还可以尝试不用 `find_MAP()` 函数，结果的差别有多大？提示：这里采样的模型非常简单，记住这个练习，后面在其他章节中我们还会在更复杂的模型上进行尝试。

（7）使用自己的数据。用你自己感兴趣的数据重新跑一边本章的内容，这个练习适用于本书剩余的所有章节。

（8）阅读 PyMC3 文档中的煤矿灾变模型：http://pymc-devs.github.io/pymc3/notebooks/getting_started.html#Case-study-2:-Coal-mining-disasters，试着自己实现并运行该模型。

除了每章最后的练习之外，你还可以将已经学到的内容应用到你感兴趣的问题上。也许，你需要重新定义你的问题，或者需要扩展或者修改你已经学到的模型。试着修改模型，如果你觉得这个任务已经超出了你实际掌握的部分，那么先将问题记下来，等读完本书其余章节后再回过头来重新思考这些问题。如果最后本书仍然无法解决你的问题，那么你可以查看下 PyMC3 的例子（http://pymc-devs.github.io/pymc3/examples.html），或者在 PyMC3 的论坛（https://discourse.pymc.io）上提问。

第 3 章 多参和分层模型

前面两章中，我们学习了贝叶斯方法的核心思想以及如何用 PyMC3 进行贝叶斯推断。如果我们想要构建任意复杂的模型（肯定会的），就必须学会构建多参模型。几乎所有我们可能感兴趣去建模的问题都需要不止一个参数，而且，在许多真实世界的问题中，某些参数可能会依赖于其他参数，这类关系可以通过分层贝叶斯模型优雅地建模。这些概念都非常重要，本书剩余部分将会反复进行回顾。

本章，我们会讨论以下主题：

- 冗余参数和边缘概率分布；
- 高斯模型；
- 存在异常值时的鲁棒估计；
- 比较不同的组以及效应值；
- 分层模型和收缩（shrinkage）。

3.1 冗余参数和边缘概率分布

尽管大多数有意思的模型都是多参的，但事实上，并非构建模型的所有参数都是我们直接感兴趣的。有时候，增加一个参数只是出于构建模型的完整性，并非因为我们真的关心这个参数。有时候我们需要估计出一个高斯分布的均值来回答某个重要问题，对于这样一个模型，除非我们知道该分布的标准差，否则在预测该分布均值的同时还需要将标准差预测出来，尽管我们对标准差并不感兴趣。我们把构建模型中并不感兴趣但却必须有的参数称为冗余参数。在贝叶斯框架下，所有未知量都是同等对待的，因而一个参数是否是冗余参数与实际的问题相关，而不是与参数本身、模型或者是推断过程相关。

此刻你可能在想，构建模型的过程中居然需要用到并不感兴趣的参数，这简直就是个负担。恰恰相反，模型中包含这些不感兴趣的参数之后，我们可以利用这类参数包含的不确定性自动传播到最后目标参数估计的结果中。在许多问题中，我们需要将一些测量值转化成感兴趣的量；例如在**磁共振成像**（Magnetic Resonance Imaging，MRI）中，我们将被特定原子核（主要是氢原子）发散和吸收的无线电频率转化成一个人内部的影像。这类转换通常需要用到冗余参数，大多数情况下，人们会将其设为某些预先校准的值，或者是某些经验值，又或许是根据某个勉强适用的经验性准则得到的值，不过贝叶斯统计可以估计出这些冗余参数（及其不确定性）。

对于两个参数的模型，我们可以把贝叶斯模型写成如下形式：

$$p(\theta_1,\theta_2 \mid y) \propto p(y \mid \theta_1,\theta_2)\,p(\theta_1,\theta_2)$$

只需要继续增加 θ，我们就可以把上式扩展到超过两个参数的情况。我们假设 θ_1 和 θ_2 是标量（数字）而不是向量。上面的式子与前面章节中看到的式子的一个不同点是：现在我们得到的是一个用 θ_1 和 θ_2 两个参数的联合分布表示的二维后验分布。我们假设 θ_2 是我们问题中的一个冗余参数，我们如何只用 θ_1 来表示后验呢？只需要计算后验分布对 θ_2 的边缘分布即可：

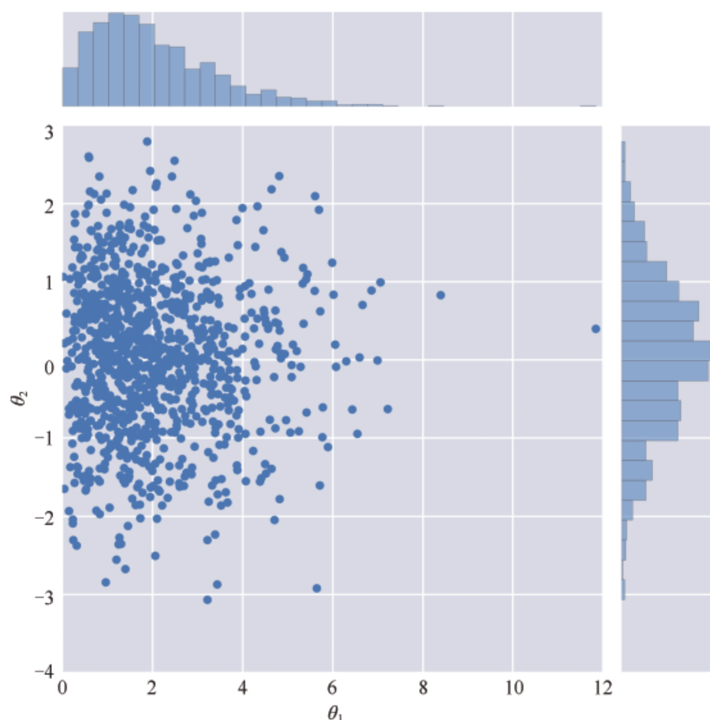
$$p(\theta_1, y) = \int p(\theta_1,\theta_2 \mid y)\,\mathrm{d}\theta_2$$

也就是说，我们将后验对 θ_2 求积分，从而将后验表示成只包含 θ_1 的项，同时隐式地考虑了 θ_2 的不确定性。对离散变量而言，积分就变成了求和。下图中，中央的点表示 θ_1 和 θ_2 的联合分布，上侧和右侧分别表示 θ_1 和 θ_2 的边缘分布。

因此，每当我们听到参数 x 的边缘分布时，我们就要联想到 x 在其他参数取遍所有可能分布后得到的平均分布。

边缘化不仅仅是从多峰分布中得到较低维度的一个切片的方法，而且还有利于简化数学分析和计算分析。有时候，我们可以在计算后验之前先对冗余参数从理论上边缘化，在第 7 章中会见到相关的例子。

通过仿真得到后验（例如用 PyMC3）的一个好处是，我们会对模型中的每个参数得到一个单独的向量，也就是说，参数已经边缘化了。



3.2　随处可见的高斯分布

前面我们用 beta- 二项分布模型介绍了贝叶斯的思想，该模型很简单。另外一个非常简单的模型是高斯分布或者叫正态分布。从数学的角度来看，高斯分布非常受欢迎的原因是它处理起来非常简单，例如，高斯分布的平均值的共轭先验还是高斯分布。此外，许多现象都可以用高斯分布来近似；本质上来说，每当我们测量某种均值时，只要采样的样本量足够大，观测值的分布就会呈现高斯分布。至于这种近似什么时候是对的，什么时候是错的，可以了解下**中心极限定理**（Central Limit Theorem，CLT），去搜索下这个统计学的核心概念。这里举一个例子，身高（以及其他描述人的特征）是受到基因和许多环境因素影响的，因而我们观测到的成年人的身高符合高斯分布。不过事实上，我们得到的其实是一个双峰分布，男人和女人的身高分布重叠在了一起。总的来说，高斯分布用起来很简单，而且自然界中随处可见，这也是为什么你了解或者听说过的许多统计方法都基于高斯分布。学习如何构建这类模型非常重要，此外，学会如何放宽正态分布

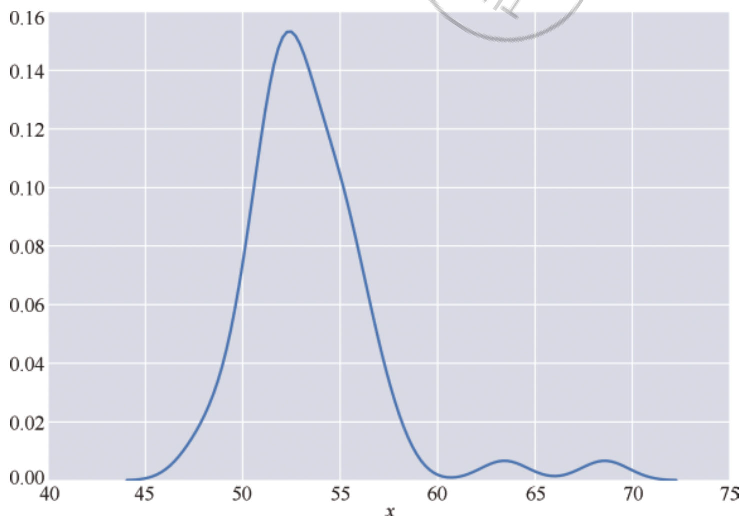
的假设也同等重要，这一点在贝叶斯框架中利用 PyMC3 之类的现代计算工具很容易处理。

3.2.1 高斯推断

下面的例子与核磁共振中的实验有关，核磁共振是一种研究分子和生物（毕竟生物也是由分子构成的）的技术。下面这组数据，可能来自一群人身高的测量值、回家的平均时间、从超市买回来橙子的重量、大壁虎的伴侣个数或者任何可以用高斯分布近似的测量值。在这个例子中，我们有 48 个测量值：

```
data = np.array([51.06, 55.12, 53.73, 50.24, 52.05, 56.40, 48.45, 52.34,
55.65, 51.49, 51.86, 63.43, 53.00, 56.09, 51.93, 52.31, 52.33, 57.48, 57.44,
55.14, 53.93, 54.62, 56.09, 68.58, 51.36, 55.47, 50.73, 51.94, 54.95, 50.39,
52.91, 51.5, 52.68, 47.72, 49.73, 51.82, 54.99, 52.84, 53.19, 54.52, 51.46,
53.73, 51.61, 49.81, 52.42, 54.3, 53.84, 53.16])
```

下图显示了上面的数据集，看起来有点像高斯分布，除了有两个点偏离了均值。

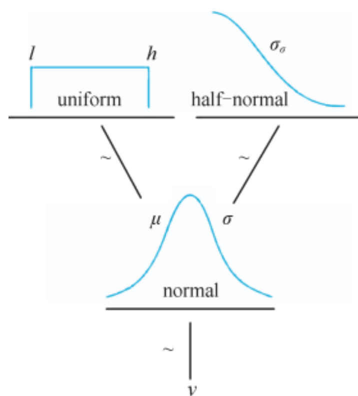


暂且先不考虑偏离均值的那两个点，假设以上分布就是高斯分布。由于我们不知道均值和方差，需要先对这两个变量设置先验。然后，顺理成章地得到

如下模型：

$$\mu \sim Uniform(l,h)$$
$$\sigma \sim HalfNormal(\sigma_\sigma)$$
$$y \sim Normal(\mu,\sigma)$$

其中，μ 来自上下界分别为 l 和 h 的均匀分布，σ 来自标准差为 σ_σ 的半正态分布。半正态分布和普通正态分布很像，不过只包含正数，看起来就好像将普通的正态分布沿着均值对折了。最后，在我们的模型中，数据 y 来自参数分别为 μ 和 σ 的正态分布，我们可以用 Kruschke 风格的图将其画出来：



如果不知道 μ 和 σ 的值，可以通过先验来表示该未知信息。例如，可以将均匀分布的上下界分别设为（l=40, h=75），这个范围要比数据本身的范围稍大一些。或者，可以根据我们的先验知识设得更广一些，比如我们知道这类观测值不可能小于 0 或者大于 100，因而可以将均匀先验的参数设为（l=0, h=100）。对于半正态分布而言，我们可以把 σ_σ 的值设为 10，该值相对于数据的分布而言算是较大的。利用 PyMC3，我们可以将模型表示如下：

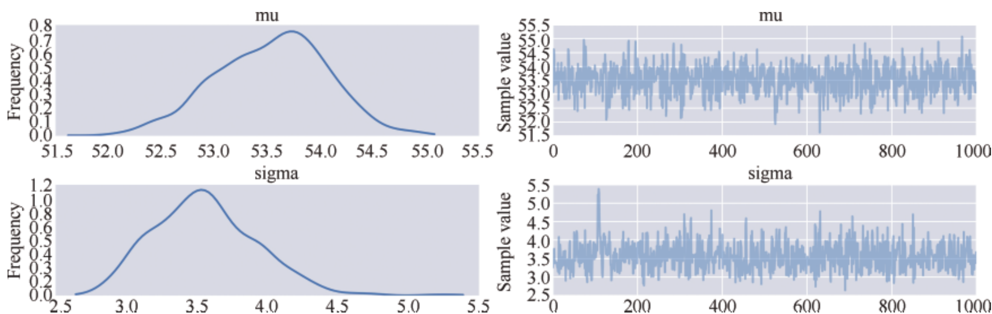
```
with pm.Model() as model_g:
    mu = pm.Uniform('mu', 40, 75)
    sigma = pm.HalfNormal('sigma', sd=10)
    y = pm.Normal('y', mu=mu, sd=sigma, observed=data)

    trace_g = pm.sample(1100)
```

traceplot 看起来很正常，我们可以接着继续分析，当然你也可以抱着怀

疑的心态跑一遍前面第 2 章提到的诊断测试。可以看到 traceplot 返回的图有两行，每行表示一个参数。这些都是边缘分布，要记住后验是二维的。

```
chain_g = trace_g[100:]
pm.traceplot(chain_g)
```



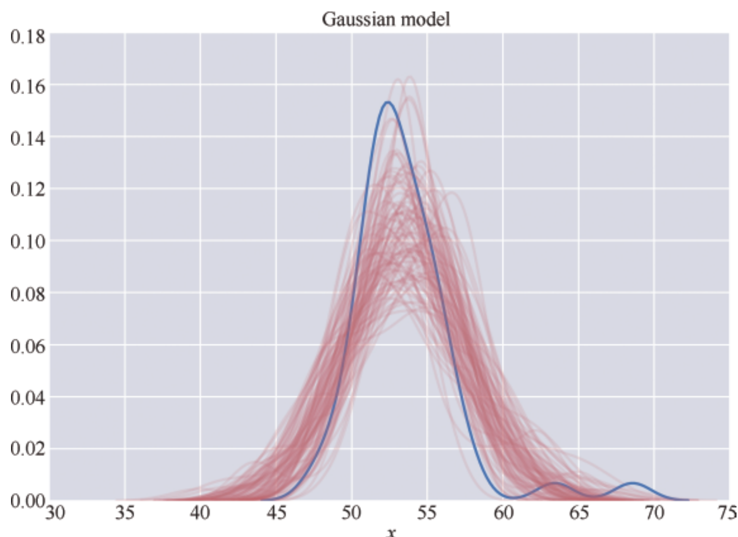
这里将参数的总结打印出来，后面会用到。

```
pm.df_summary(chain_g)
```

	mean	sd	mc_error	hpd_2.5	hpd_97.5
mu	53.51	0.53	0.02	52.56	54.54
sigma	3.55	0.38	0.01	2.86	4.32

现在我们得到了后验，可以将其用于模拟数据，然后就可以检查模拟数据与观测数据是否一致了。在第 1 章中，我们将这种检查称作后验预测检查，因为我们先通过后验做出预测，然后用这些预测来检查模型。利用 PyMC3 中的 sample_ppc() 函数可以很容易地从后验中得到预测值。下面的代码中我们从后验中生成了 100 组预测值，每组预测值的大小与观测数据的维度一致。注意我们需要将迹和模型传给 sample_ppc()，其他参数是可选的。

```
y_pred = pm.sample_ppc(chain_g, 100, model_g, size=len(data))
sns.kdeplot(data, c='b')
for i in y_pred['y']:
    sns.kdeplot(i, c='r', alpha=0.1)
plt.xlim(35, 75)
plt.title('Gaussian model', fontsize=16)
plt.xlabel('$x$', fontsize=16)
```



上图中，蓝色的线是观测数据的 KDE，半透明的红色的线是 100 组从后验中采样出来的预测值的 KDE。可以看出，采样值的均值要稍稍偏右一些，而且采样值的变化相比原始的观测值也更大一些。接下来的内容中，我们会逐步优化模型，最终得到一个与观测数据更吻合的后验分布。

3.2.2　鲁棒推断

对于前面的模型，你可能有点疑惑，我们假设数据的分布是高斯分布，但是在数据的末端却有两个数据点，这看起来不太像高斯分布。高斯分布的两端随着距离均值越远，值会迅速下降。对于我们这里的数据而言，模型中的高斯分布看到右端的那两个点时会感到很惊讶，于是会向右侧靠近，从而使得其标准差也上升了。可以看出，这两个点对于高斯分布的参数有着举足轻重的作用。那么可以采取什么措施改进呢？一个做法是将这两个点看做是异常点并将其从观测值中剔除，因为这两个值有可能是因为仪器异常或者人为疏忽导致的。有时候我们也许能够直接矫正这些点，因为有可能只是我们处理数据时的代码出了些问题，但更多时候，我们希望能够根据某种异常值的处理规则自动消除这些异常点，其中的两个规则如下：

- 所有超出 1.5 倍 4 分位范围的数据都是异常值；

- 所有超出观测数据两倍标准差的都是异常值。

t 分布

除了利用以上规则改变原始数据之外，我们还可以修改模型。通常，按照贝叶斯的思想，我们更倾向于通过使用不同的先验或者似然将假设编码到模型中，而不是直接使用一些先验准则（例如前面这些剔除异常值的准则）。

一个用来解决异常值的非常有用的方法是：将高斯分布替换成 t 分布。t 分布有 3 个参数：均值、尺度（与标准差类似）和自由度（通常用 v 表示，取值范围为 [0,¥]）。根据 Kruschke 的命名方式，我们将 v 称为正态参数，这是因为该参数决定了 t 分布与正态分布的相似程度。对于 v=1 的情况，t 分布的尾部要比高斯分布更重，在不同的领域也称柯西分布或者洛伦兹分布，这里尾部更重的意思是：相比高斯分布，我们更有可能观测到偏离均值的点，换句话说，该分布并不像高斯分布那样聚集在均值附近。举例来说，柯西分布 95% 的点都分布在 −12.7 ～ 12.7，而对于（标准差为 1 的）高斯分布，对应的区间为 −1.96 ～ 1.96。此外，当正态参数 v 趋近于无穷大时，我们就会得到高斯分布（你不可能比正态分布还正态对吧？）。t 分布一个有意思的特性是：当 $v \leqslant 1$ 时，该分布没有准确定义的均值。当然，实际中从 t 分布得到的采样不过是一些数字，因而总是可以算出经验性的均值来，不过理论上还没有一个准确定义的均值。直观上可以这么理解：t 分布的尾部很重，因而我们得到的采样值很可能是实轴上的任意一点，所以只要不停地采样，我们永远也无法得到一个固定值。你可以尝试多次运行下面的代码（或者将参数 `df` 换成一个更大的值，比如 100）：

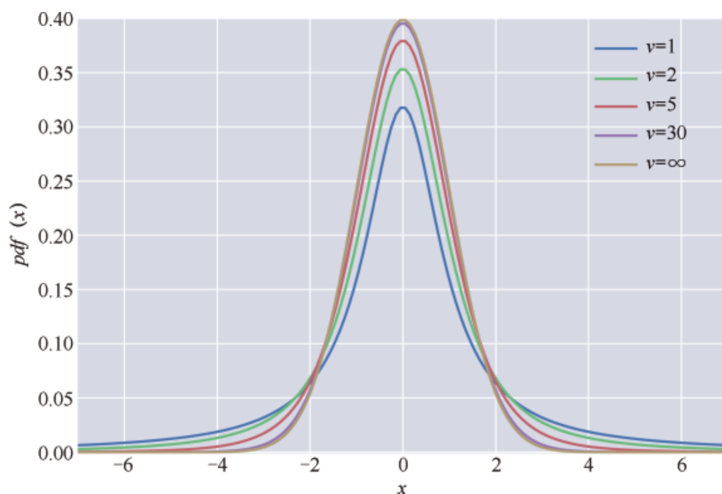
```
np.mean(stats.t(loc=0, scale=1, df=1).rvs(100))
```

类似地，只有当 v>2 时，分布的方差才有明确定义，因此，需要注意 t 分布的尺度与标准差不是同一个概念。对于 $v \leqslant 2$ 的分布，方差并没有明确定义，因而也没有明确定义的标准差。当 v 趋向于无穷大时，尺度趋近于标准差。

```
x_values = np.linspace(-10, 10, 200)
for df in [1, 2, 5, 30]:
    distri = stats.t(df)
    x_pdf = distri.pdf(x_values)
```

```
    plt.plot(x_values, x_pdf, label=r'$\nu$ = {}'.format(df))

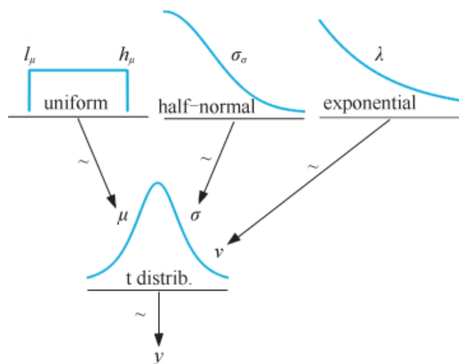
x_pdf = stats.norm.pdf(x_values)
plt.plot(x_values, x_pdf, label=r'$\nu = \infty$')
plt.xlabel('x')
plt.ylabel('p(x)', rotation=0)
plt.legend(loc=0, fontsize=14)
plt.xlim(-7, 7)
```



利用 t 分布将模型表示成如下形式：

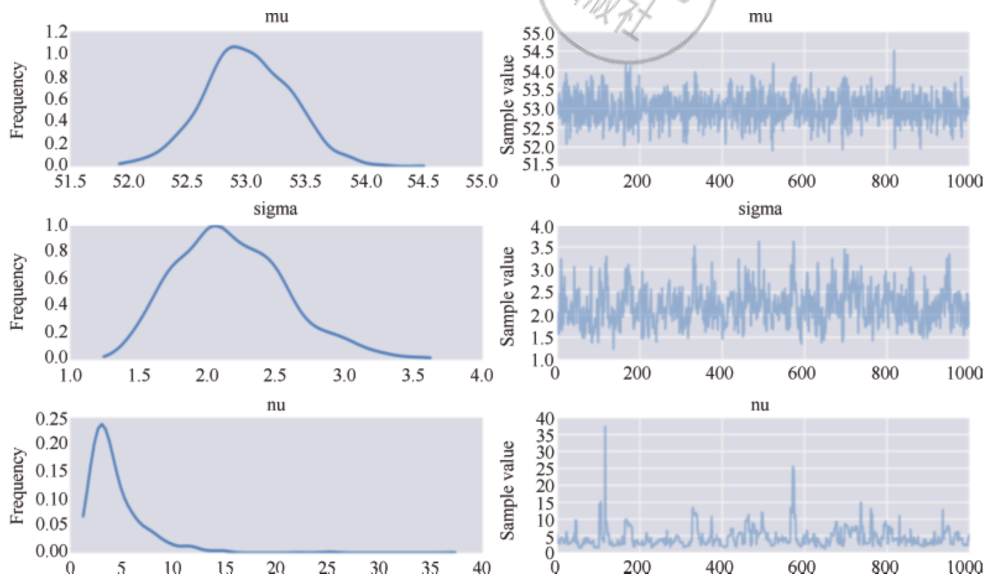
$$\mu \sim Uniform(l,h)$$
$$\sigma \sim HalfNormal(\sigma_h)$$
$$v \sim Exponential(\lambda)$$
$$y \sim Student(\mu,\sigma,v)$$

上面这个模型与前面的高斯模型的主要区别是：现在似然是 t 分布，由于 t 分布多了一个新的参数，我们需要为其增加一个先验。这里用了一个均值为 30 的指数分布。上图可以看出，t 分布看起来很像高斯分布（尽管其实并不一样）。v 值较小的分布更分散，因而，均值为 30 的指数分布是一个很弱的先验，认为正态参数 v 在 30 附近，不过也可以很容易地将其调大或调小。从图像上看，我们的模型表示如下：



同样，PyMC3 让我们只需要几行代码便可修改模型。唯一需要注意的是，PyMC3 中指数分布的参数用的是分布均值的倒数。

```
with pm.Model() as model_t:
    mu = pm.Uniform('mu', 40, 75)
    sigma = pm.HalfNormal('sigma', sd=10)
    nu = pm.Exponential('nu', 1/30)
    y = pm.StudentT('y', mu=mu, sd=sigma, nu=nu, observed=data)
    trace_t = pm.sample(1100)
chain_t = trace_t[100:]
pm.trace_plot(chain_t)
```

现在用 summary 函数将迹的总结打印出来并与前面的结果对比。继续深入阅读之前，花点时间对比分析下两组不同的结果，你能发现什么有趣的地方吗？

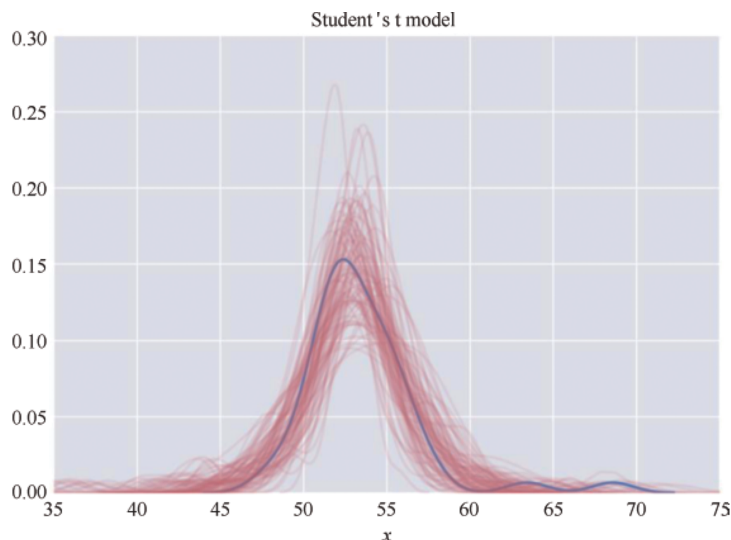
```
pm.df_summary(chain_t)
```

	mean	sd	mc_error	hpd_2.5	hpd_97.5
mu	52.99	0.38	0.01	52.28	53.81
sigma	2.15	0.39	0.02	1.42	2.97
nu	4.13	2.78	0.19	1.19	8.54

可以看到，两个模型对 μ 的估计比较接近，只相差 0.5 左右，而 σ 的估计则从 3.5 变成了 2.1，这正是因为 t 分布对于偏离均值的点所赋予的权重较小所致。此外还可以看到，μ 的值接近 4，也就是说，该分布并不太像高斯分布，而是更接近重尾分布。

接下来我们对 t 分布模型做后验检查，并将其与高斯分布对比：

```
y_pred = pm.sample_ppc(chain_t, 100, model_t, size=len(data))
sns.kdeplot(data, c='b')
for i in y_pred['y']:
    sns.kdeplot(i, c='r', alpha=0.1)
plt.xlim(35, 75)
plt.title("Student's t model", fontsize=16)
plt.xlabel('$x$', fontsize=16)
```



可以看到，使用 t 分布之后，从分布的峰值和形状来看，模型的预测值与观测数据更吻合了（留意预测值远离观测值中心的部分）。这是因为 t 分布希望看到在偏离数据中心的两个方向上都有数据。在我们的模型中，t 分布的估计值更

鲁棒，因为异常点降低了正态参数 v 的值，从而均值和尺度更多地是从观测数据的中心估计出来的，而不是像前面高斯分布的例子中那样，均值和标准差都偏向了异常值。再强调一次，这里的尺度并不是标准差。不过，尺度这个参数确实与数据的分散程度有关，尺度越小，数据分布得越集中。此外，对应正态参数 v 大于 2 的情况，尺度的值倾向于接近去掉异常值之后的标准差。因此，粗略地讲，对于不是特别小的 v 值，我们可以将 t 分布的尺度大致看做是去掉异常值之后的数据的标准差（理论上这么说可能不太对）。

3.3　组间比较

统计分析中一个常见的任务是对不同的组进行比较，例如我们可能想知道病人对某种药的反应如何、引入某种交通法规后车祸数量是否会降低、学生对不同教学方式的表现如何等。有时候，这类问题统一归到了假设检验的框架下，其目的是得到统计学意义上的显著性。仅仅依赖于统计显著性可能会带来很多问题：一方面，统计显著性并非实际显著性；另一方面，即使是很小的作用，只要收集尽可能多的数据，都会被看做具有显著性。而且，统计显著性的核心思想往往需要计算 p 值。已经有很多文章和研究记录表明，通常，p 值会被错误地使用和解释，即使那些每天跟统计打交道的科学家也会犯错。不过在贝叶斯框架下，我们不需要计算 p 值，所以这里暂且将其放一边。毕竟，在实践中我们最想知道的是**效应值**，也就是量化估计出某种现象的强弱。

在比较不同组的数据时，人们往往会将其分为一个实验组和一个对照组（也可能超过一个实验组和对照组），例如当我们测试一个新药时，由于安慰剂效应或者某些其他原因，我们希望将使用新药的组（实验组）和不使用新药的组（对照组）进行对比。在这个例子中，我们想知道对于治疗某种疾病，使用药物对比不用药物（或者是使用安慰剂）的作用有多大。另一个有趣的问题是：与治疗某种疾病最常用的（已经被审批的）药相比，我们的药效果如何？此时，对照组不再是使用安慰剂的组，而是使用其他药物的组。从统计学上讲，使用假的对照组是一种不错的撒谎方式，例如，假设某家邪恶的乳制品公司想要卖某种糖过量的酸奶给小朋友，同时告诉他们的家长乳酸有利于免疫系统。一种支撑该说法的做法是：使用牛奶或者水作为对照组，而不是用更便宜、糖更少、市场更小的酸

奶。这么做听起来很笨，不过下次再听到有人说某种东西更坚固、更好、更快、更强时，记得问一下他对比的基准线是什么。

3.3.1 “小费”数据集

接下来，我们将使用 seaborn 中的 tips 数据集来讨论前面提到的那些思想。我们希望知道星期几对于餐馆小费数量的影响。这个例子中，实际上并没有明确的实验组和对照组之分，这只是观察性的实验，并非像前面药物测试的例子一样。如果愿意的话，我们可以任选一天（例如星期四）作为实验组，或者对照组，尽管我们并没有控制某个具体的东西。需要注意的一点是：对于观察性实验，我们没法得出某种因果关系，能得到的只是相关性。事实上，如何从数据中得出因果关系是一个非常热门的研究问题，我们会在第 4 章重新讨论这个问题。现在，我们首先用一行代码将数据导入成 Pandas 中的数据结构，如果你对 Pandas 不太熟悉，这里需要说明下，tail 函数返回数据中的最后一部分（当然你也可以用 head 函数返回前面一部分数据）。

```
tips = sns.load_dataset('tips')
tips.tail()
```

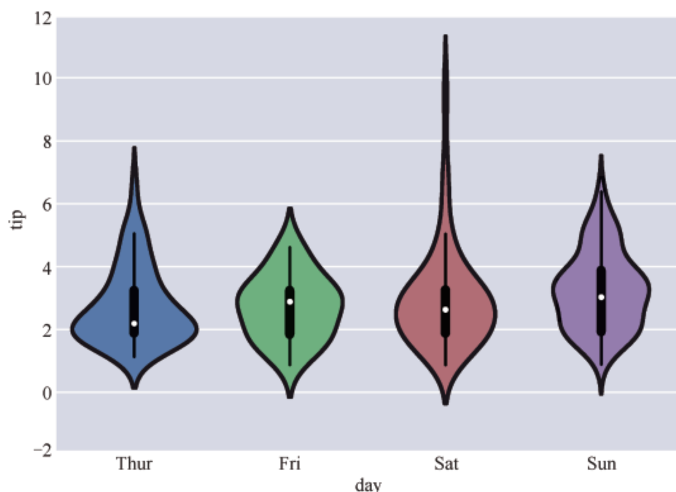
	total_bill	tip	sex	smoker	day	time	size
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

对于这个数据集，我们只关心其中的 day 和 tip 列，利用 seaborn 中的 violinplot 函数可以将其画出来：

```
sns.violinplot(x='day', y='tip', data=tips)
```

把问题简化下，我们创建两个变量：变量 y 表示 tips；变量 idx 表示分类变量的编码。也就是说，我们用数字 0、1、2、3 表示星期四、星期五、星期六和星期天。

```
y = tips['tip'].values
idx = pd.Categorical(tips['day']).codes
```

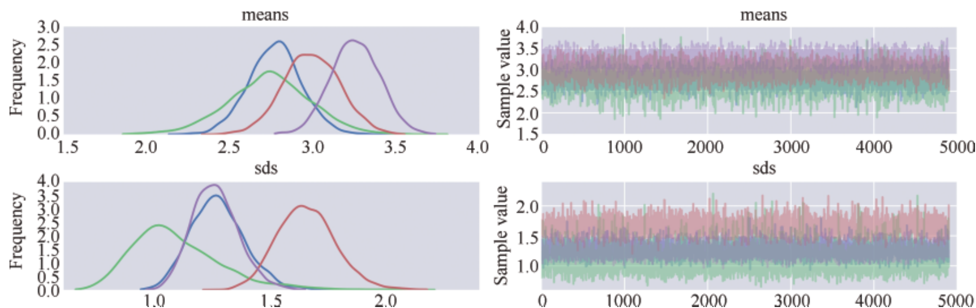


这个问题中的模型与之前的模型几乎一样，唯一的区别是 μ 和 σ 现在是一组随机向量而不再是一个标量。换句话说，每次从先验中采样的时候，我们会得到 4 个 μ 和 4 个 σ。PyMC3 的语法能够很好地适应这个场景，我们可以直接用向量的方式表示模型，而不用使用 for 循环，代码的变化相比前面的模型很小。对应先验，我们需要传一个维度变量 shape，对于似然，我们需要对 μ 和 σ 正确地进行编码，这也是为什么创建了 idx 变量。

```
with pm.Model() as comparing_groups:
    means = pm.Normal('means', mu=0, sd=10, shape=len(set(x)))
    sds = pm.HalfNormal('sds', sd=10, shape=len(set(x)))

    y = pm.Normal('y', mu=means[idx], sd=sds[idx], observed=y)

    trace_cg = pm.sample(5000)
chain_cg = trace_cg[100::]
pm.traceplot(chain_cg)
```



这里照常使用 df_summary 函数来描述估计点，同样你还可以进行诊断测试。切记贝叶斯分析返回的是（在给定数据和模型条件下）参数的完整分布，因而我们可以对后验进行进一步处理，并从中提出一些合理的问题。比如，我们可能想知道组与组之间均值差别的分布情况，下面，我们就来分析看看。

这里我们使用 PyMC3 中的 plot_posterior 函数画出后验分布，其中选取参考值（ref_val）为 0，因为我们希望将后验与 0 进行比较。下面的代码将两个变量的差画了出来，没有进行重复比较。注意这里并没有一对一的对比矩阵，只是画出了上三角部分。代码和图中最陌生的部分是 **Cohen's d** 和**概率优势**，后面会详细解释，其实它们都是对效应值的不同表示方式而已。

```
dist = dist = stats.norm()
_, ax = plt.subplots(3, 2, figsize=(16, 12))

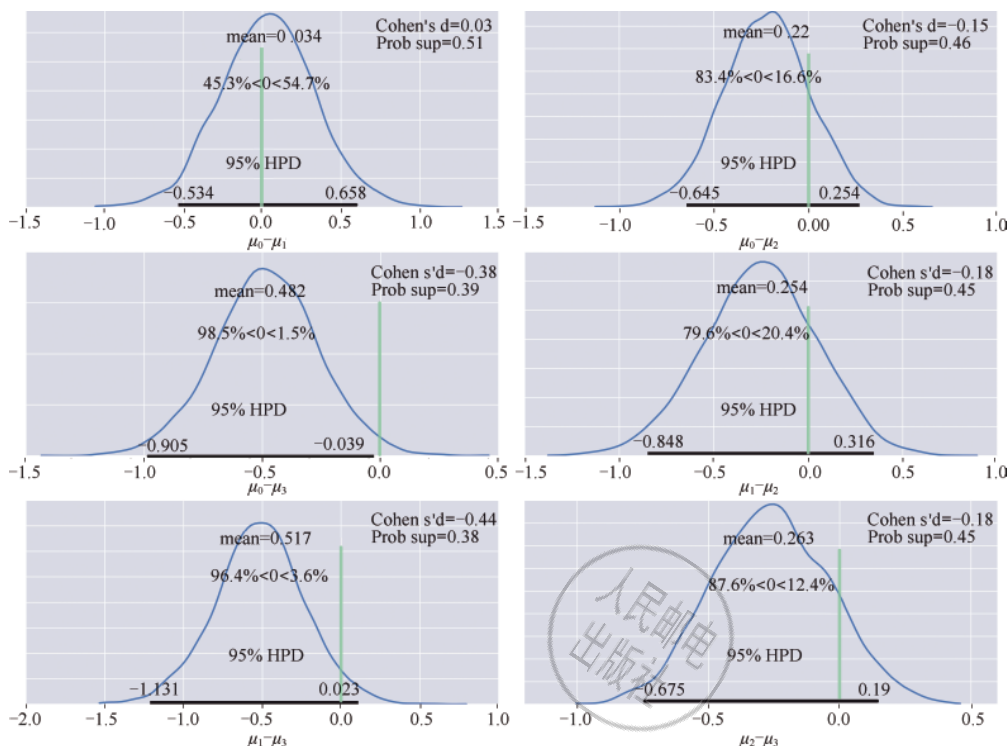
comparisons = [(i,j) for i in range(4) for j in range(i+1, 4)]
pos = [(k,l) for k in range(3) for l in (0, 1)]

for (i, j), (k,l) in zip(comparisons, pos):
    means_diff = chain_cg['means'][:,i]-chain_cg['means'][:,j]
    d_cohen = (means_diff / np.sqrt((chain_cg['sds'][:,i]**2 + chain_cg['sds']
[:,j]**2) / 2)).mean()
    ps = dist.cdf(d_cohen/(2**0.5))

    pm.plot_posterior(means_diff, ref_val=0, ax=ax[k, l], color='skyblue')
    ax[k, l].plot(0, label="Cohen's d = {:.2f}\nProb sup = {:.2f}".format(d_
cohen, ps) ,alpha=0)
    ax[k, l].set_xlabel('$\mu_{}-\mu_{}$'.format(i, j), fontsize=18)
    ax[k,l ].legend(loc=0, fontsize=14)
```

前面的例子中，一种解释结果的方式是将参考值与 HPD 区间进行比较。只有一种情况下 95%HPD 没有包含 0（我们的参考值），即星期四与星期天的对比。对于所有其他情况，我们没法得出两者的区别为 0 的结论（根据 HPD 区间与参考值的重叠性准则）。但是即便如此，平均下来 0.5 美元的小费差别是否足够大了呢？这个差别是否大到让人们牺牲星期日陪家人或者朋友的时间去工作呢？是否大到就应该这 4 天都给相同的小费而且男服务员和女服务员的小费一模一样呢？诸如此类的问题很难用统计学来回答，只能从统计学中找到些启发。 描述效应值的方式有好几种，我们接下来将学习其中的两个：Cohen's d 和

概率优势。



3.3.2　Cohen's d

Cohen's d 是一种用来描述效应值的常见方式：

$$\frac{\mu_1-\mu_2}{\sqrt{\frac{\sigma_1^2+\sigma_2^2}{2}}}$$

也就是说，效应值是在考虑不同组的标准差的情况下，均值的差异。在前面的代码中，我们根据估计值的均值和标准差算出了 Cohen's d 的值，因而我们可以添加对 Cohen's d 的描述而不仅仅是均值。

比较各组数据的时候，很重要的一点是要考虑组内的波动性（比如标准差）。一组数据相比另一组数据变化了 x 个单位，可能是每个点都整体变化了 x 个单位，也可能是其中一半的数据没有变化而另外一半数据变化了 $2x$，还可能是其他组

合。根据 Cohen’s d 得到的效应值可以看做是 Z-score，因而 Cohen’s d 为 0.5 可以解释为一组数据相比另一组数据点的差别是 0.5 倍的标准差。使用 Cohen’s d 的一个问题是不太好解释，我们需要根据具体的问题来说明该值是太大、太小或者是适中。当然，我们可以从实践中得到些经验，不过更多地还是依赖于具体问题。假如说我们对同一类问题做了一些分析，然后得到 Cohen’s d 的值约为 1，这时如果得到另外一个 Cohen’s d 的值（比如说 2），那么我们很可能有了重要发现（也可能是某个地方弄错了）。在 http://rpsychologist.com/d3/cohend 这个网页中，你可以探索一下不同 Cohen’s d 的值都长什么样，此外，在这个网页中还可以看到一些描述效应值的其他方式，其中某些方式可能更直观（比如概率优势）。

3.3.3 概率优势

概率优势是表示效应值的另一种方式，描述的是从一组数据中取出的一个点大于从另外一组中取出的点的概率。假设两个组中数据的分布都是正态分布，我们可以通过以下表达式从 Cohen’s d 中得到概率优势：

$$ps = \phi\left(\frac{\delta}{\sqrt{2}}\right)$$

其中，ϕ是累计正态分布，δ 是 Cohen’s d。我们可以算出概率优势的点估计（通常列出的是该值），也可以计算出概率优势的分布。注意到，我们可以用该式根据 Cohen’s d 来计算概率优势，或者，我们可以直接将其从后验中计算出来（查看练习部分）。这正是使用 MCMC 方法的一个很大好处。一旦我们从后验中得到了采样，我们就可以算出某些值（比如概率优势等）而不必依赖于具体的分布假设。

3.4 分层模型

假设我们想要分析一个城市的水质，然后将城市分成了多个相邻（或者水文学上）的区域。我们可以用如下两种方法进行分析：

- 分别对每个区域单独进行估计；

■ 将所有数据都混合在一起，把整个城市看做一个整体进行估计。

两种方式都是合理的，具体使用哪种取决于我们想知道什么。如果我们想了解具体的细节，那么可以采用第 1 种方式，因为假如对数据进一步做了一些平均处理，那么一些细节就不太容易看出来了。采用第 2 种方式则可以将数据都聚在一起，得到一个更大的样本集，从而得出更准确的估计。两种方式都有其合理性，不过我们还可以找到些中间方案。我们可以在对相邻区域的水质进行评估的同时对整个城市的水质进行评估，这类模型称为层次化模型或者分层模型，这么称呼的原因是我们对数据采用了一种层次化（或者是分层）的建模方式。

那么如何构建分层模型呢？简单说，就是在先验之上使用一个共享先验。也就是说，我们不再固定先验参数，而是直接从数据中将其估计出来。这类更高层的先验通常称为超先验（hyper-prior），它们的参数称为超参数，这里“超”在希腊语中是“在某某之上”的意思。当然，还可以在超先验之上再增加先验，做到尽可能分层。问题是这么做会使得模型变得相当复杂而难以理解，而且除非问题确实需要更复杂的结构，增加分层对于做推断并没有更大帮助，相反，我们会陷入超参和超先验的混乱中而无法对其做出任何有意义的解释，从而降低模型的可解释性。毕竟，我们建模的首要目的是理解数据。

为了更好地解释分层模型中的主要概念，我们以本节开头提到的水质模型作为例子，使用一些构造的数据来讲解。假设我们从同一个城市的 3 个不同水域得到了含铅量的采样值：其中高于世界卫生组织（World Health Organization,WHO）标准的值标记为 0；低于标准的值标记为 1。这个例子只是用来教学，实际中，我们会使用铅含量的连续值，并且可能会分成更多组。不过，对我们来说，这个例子足够用来揭示多层模型的细节了。

我们通过以下代码合成数据：

```
N_samples =  [30, 30, 30]
G_samples =  [18, 18, 18]

group_idx = np.repeat(np.arange(len(N_samples)), N_samples)
data = []
for i in range(0, len(N_samples)):
    data.extend(np.repeat([1, 0], [G_samples[i], N_samples[i]-G_samples[i]]))
```

这里进行了一个仿真实验，分别对 3 个组进行了一定次数的采样。我们将每

组的采样总数放在列表 `N_samples` 中，用列表 `G_samples` 记录每组中合格的采样值。剩下的代码用于生成 0、1 数据。

模型本质上还是抛硬币问题中的那个模型，不同之处在于需要指定影响 beta 先验的超先验：

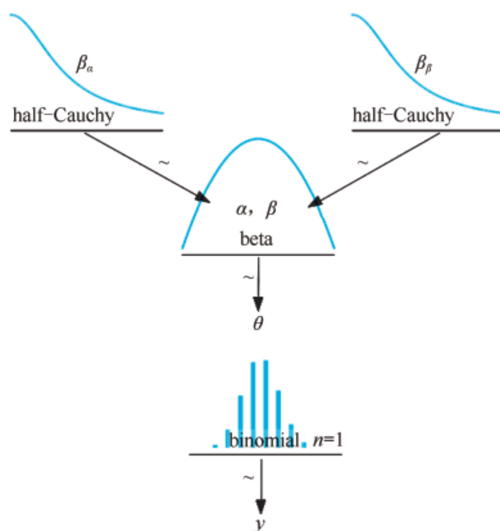
$$\alpha \sim HalfCauchy\left(\beta_\alpha\right)$$

$$\beta \sim HalfCauchy\left(\beta_\beta\right)$$

$$\theta \sim Beta\left(\alpha, \beta\right)$$

$$y \sim Bern\left(\theta\right)$$

使用 Kruschke 图，可以清楚地看到新的模型相比原来的模型多了一层。

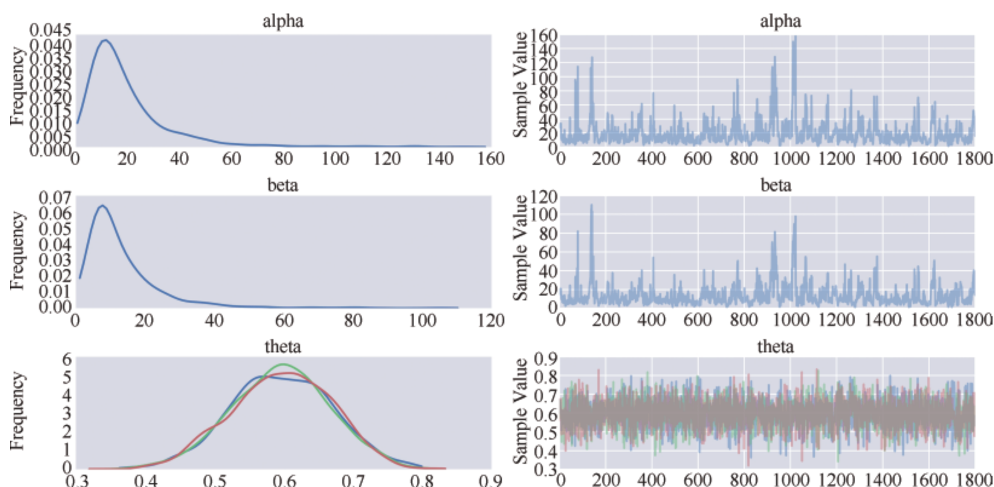



```
with pm.Model() as model_h:
    alpha = pm.HalfCauchy('alpha', beta=10)
    beta = pm.HalfCauchy('beta', beta=10)

    theta = pm.Beta('theta', alpha, beta, shape=len(N_samples))

    y = pm.Bernoulli('y', p=theta[group_idx], observed=data)

    trace_j = pm.sample(2000)
chain_h = trace_h[200:]
pm.traceplot(chain_h)
```



3.4.1　收缩

现在和我一起做个简单的实验。我需要你输出模型总结并且将结果保存下来待会用，然后再分别重新运行模型两次，其中一次将所有的 `G_samples` 都设为 3，另外一次将 `G_samples` 设为 [18,3,3]，并且每次都记录下模型的总结。继续阅读之前，先想想这个实验的结果会是什么。重点关注每次实验中 θ 的均值。根据前两次模型的运行结果，你能猜出第 3 种情况下的结果吗？

如果将结果汇总在表格中，会得到类似如下的值（注意由于 NUTS 采样方法的随机性，结果可能会有小幅波动）：

G_samples	Theta(mean)
18,18,18	0.6,0.6,0.6
3,3,3	0.1,0.1,0.1
18,3,3	0.53,0.14,0.14

表中第一行，可以看到对于 30 个样本中有 18 个正样本的情况，θ 估计值的均值为 0.6；注意现在 $\boldsymbol{\theta}$ 是一个向量，因为现在我们有 3 个组，每个组都有一个均值。第 2 行中，30 个样本中有 3 个是正样本，得到的 θ 的均值为 0.1。最后一行中的结果有点意外，θ 的均值并非是前面两组中均值的组合（比如 0.6,0.1,0.1），而是 0.53,0.14,0.14。为什么呢？是模型收敛的问题还是模型选型出

了问题？都不是，是我们的估计结果趋向了整体的均值。事实上，这正是我们模型预期的结果，在设置了超先验后，我们直接从数据中估计（beta）先验，每个组的估计都受到了其他组的估计值的影响，同时也影响着其他组的估计值。换句话说，所有组都通过超先验共享了部分信息，从而看到一种称为**收缩**的现象，其效果相当于对数据做了部分“池化”（pooling），我们既不是在对数据分组建模，也不是将数据看做一个大组在建模，而是介于二者之间，其结果之一就是收缩效应。

收缩有利于更稳定的推断。这一点和前面讨论过的 t 分布与异常点的关系很像。使用重尾分布之后的模型对于偏离均值的异常点表现得更鲁棒（更不受其影响）。引入超先验后，我们在更高的层次上进行推断，从而得到一个更“保守”的模型（这可能是我第一次将“保守”这个词当做褒义词），更少地受到每个组中极限值的影响。举例来说，假设我们在某个相邻区域得到了一组不同数量的采样值；采样数量越小就越容易得到错误的结果。极限情况下，假设在这片区域只有一个采样值，你可能恰好是从这片区域的某个铅管中得到的采样值，或者，有可能恰好是从 PVC 管道中得到的采样值，从而可能导致你对这片区域的水质高估或者低估。在多层模型中，估计出错的情况可以通过其他组提供的信息进行改善。当然，更大的采样值同样能达到类似的效果，不过大多数情况下这并不是个候选方案。

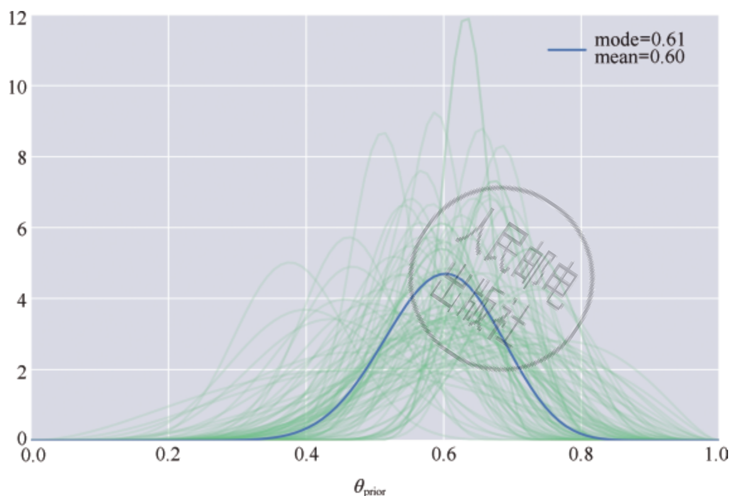
显然，收缩的程度取决于数据，数量更大的组会对其他数量较小的组造成更大的影响。如果大多数组都比较相似，而其中某组不太一样，相似的组之间会共享这种相似性，从而强化共同的估计值，并拉近表现不太一样的那一组的估计值，前面的例子中也已经体现了这一点。此外，超先验也对收缩的程度有影响。如果我们对所有组的整体分布有一些可以信赖的先验信息，那么可以将其加入到模型中并将收缩程度调整到一个合理的值。我们完全可以只用两个组来构建层级化模型，不过通常我们更倾向于使用多个组。直观上的原因是，收缩其实是将每个组看成了一个数据点，然后我们在组这一层估计标准差。通常我们不会太相信点数较少的估计值，除非对估计值有很强的先验，这一点对层次化模型也适用。

你可能对估计到的先验分布比较感兴趣，以下是将其表示出来的一种方式：

```
x = np.linspace(0, 1, 100)
for i in np.random.randint(0, len(chain_h), size=100):
    pdf = stats.beta(chain_h['alpha'][i], chain_h['beta'][i]).pdf(x)
    plt.plot(x, pdf,  'g', alpha=0.05)

dist = stats.beta(chain_h['alpha'].mean(), chain_h['beta'].mean())
pdf = dist.pdf(x)
mode = x[np.argmax(pdf)]
mean = dist.moment(1)
plt.plot(x, pdf, label='mode = {:.2f}\nmean = {:.2f}'.format(mode, mean))

plt.legend(fontsize=14)
plt.xlabel(r'$\theta_{prior}$', fontsize=16)
```



套用 Python 之禅的说法，我们可以说，分层模型是一种绝妙的理念，我们应当多加利用！[①] 在接下来的章节中，我们会继续构建分层模型并学习如何构建更好的模型。在第 6 章模型比较部分，我们还会详细讨论构建模型过程中的过拟合和欠拟合问题。

3.5　总结

这章我们使用多个参数扩展了构建模型的能力，在 PyMC3 的帮助下这非常

① Python 之禅的最后一句话是：“命名空间是一种绝妙的理念，我们应当多加利用！”。——译者注

容易实现。例如，从后验中计算边缘分布只需要从迹中选出相应的部分即可。此外，我们用几个例子学习了如何从后验分布中提取我们感兴趣的量，比如合成数据，或者是提取能够更好地解释数据的量。起初我们使用的是高斯模型，这主要是因为高斯模型是数据分析的支柱之一，不过在包含异常值的数据上应用高斯分布时遇到了一些问题，然后我们学习了使用 t 分布对数据的正态性假设做一些放松，从而了解了鲁棒模型的概念以及如何改进一个模型使其适用于具体的问题。对不同数据组进行比较是数据分析中的一个常见任务，对此我们使用了高斯模型，并且讨论了一些量化指标来衡量不同组之间数据的区别。最后，我们学到的核心概念之一是：分层模型，或者说是如何结构化地解决问题从而更好地做推断，并且通过部分池化不同组的信息收缩估计值。

下一章我们将学习线性模型以及如何利用线性模型解释数据。

3.6 深入阅读

- 《Doing Bayesian Data Analysis, Second Edition》中的第 9 章
- 《Statistical Rethinking》中的第 12 章
- 《Bayesian Data Analysis, Third Edition》中的第 5 章
- 阅读完本书第 4 章后，记得读一下 PyMC3 的文档中有关 GLM 的多层模型以及 Rugby 的例子

3.7 练习

（1）对于本章的第一个模型，将高斯分布的先验均值修改为一个经验性均值，用几个对应的标准差多跑几遍，观察推断过程对这些变化的鲁棒性 / 敏感性如何。你觉得用一个没有限制上下界的高斯分布对有上下界的数据建模的效果怎样？记住我们说过数据不可能大于 100 或者小于 0。

（2）利用第一个例子中的数据，分别在包含和不包含异常值情况下，计算出经验均值和标准差。将结果与使用高斯分布和 t 分布的贝叶斯估计进行比较，增加更多异常值并重复该过程。

（3）修改小费例子中的模型，使其对于异常点更鲁棒。分别尝试对所有组使用一个共享的 v 和单独为每个组设置一个 v，最后对这 3 个模型进行后验预测检查。

（4）直接从后验中计算出概率优势（先不要计算 Cohen’s d），你可以用 `sample_ppc()` 函数从每个组中获取一个采样值。对比这样做与基于正态假设的计算是否不同？并对结果做出解释。

（5）重复水质对比的例子，不过不用多层模型，而是使用一个均匀分布（比如 $Beta(\alpha=1, \beta=1)$）。比较两种模型的结果。

（6）在小费的例子中，对一个星期中的不同天使用部分池化操作，构建一个多层模型，将结果与不使用多层模型的结果进行对比。

（7）重复本章中的所有例子，用 `findMAP()` 函数的返回值来初始化采样。看是否能得到相同的推断结果。同时看一下 `find_MAP()` 函数对退化过程的数量以及推断的速度有什么影响。

（8）对所有模型进行诊断测试并采取相应措施，比如，如果有必要，增加采样次数。

（9）对本章中的至少一个模型使用你自己的数据并运行。牢记第 1 章中提到的构建模型的 3 个步骤。

第 4 章 利用线性回归模型理解并预测数据

这一章我们将学习统计学和机器学习中应用最广泛的模型之一：线性回归模型。这个模型除了本身非常有用之外，还可以看做是许多其他模型的基石。如果你上过统计学的课程（即使是非贝叶斯的），你可能会听说过线性回归、逻辑回归、ANOVA 和 ANCOVA 等。所有这些方法都是同一个主题的变种：线性回归模型，这也正是本章将要讨论的核心主题。

本章将涵盖以下内容：

- 线性回归模型
- 一元线性回归；
- 鲁棒线性回归；
- 多层线性回归；
- 多项式回归；
- 多元线性回归；
- 交互作用。

4.1 一元线性回归

在科学界、工业界及商业中，经常会遇到下面这类问题：我们有一些连续变量，这里连续的意思是变量可以用实数表示（或者说是浮点数），我们称之为**因变量**、**被预测的变量**或者**结果变量**。我们想要对该因变量与其他变量的依赖关系建模，这里其他变量称**自变量**、**预测变量**或者**输入变量**。自变量可以是连续的，也可以是离散的。这类问题一般可以通过线性回归建模，如果我们只有一个自变量，那么称为一元线性回归模型，如果有多个自变量，称为作多元线性回归模型。使用线性模型的一些典型场景如下。

- 对多个因素之间的关系建模，例如雨量、土壤盐渍度以及农作物生长过程中是否施肥，然后回答一些问题：比如它们之间的关系是否是线性的？关系有多强？哪个因素的影响最强？
- 找出全国平均的巧克力摄入量与诺贝尔奖得主数量之间的关系。理解为什么这二者之间的关系可能是假的。
- 根据当地天气预报中的太阳辐射预测家里（用于烧水和做饭）的燃气账单。该预测的准确性如何？

4.1.1 与机器学习的联系

按照 Kevin P. Murphy[①] 的说法，**机器学习**是一个总称，指一系列从数据中自动学习其中的隐藏规律，并用来预测未知数据，或者是在不确定的状态中做决策的方法。机器学习与统计学相互交织，不过正如 Kevin P.Murphy 在他书中所说，如果从概率的角度来看，二者之间的关系就比较清晰了。尽管这两个领域在概念上和数学上都紧密联系，不过二者之间的术语可能让这种联系显得不那么清晰。因此，在这一章中，我会介绍一些机器学习中的术语。用机器学习的行话来说，回归问题属于典型的监督学习。在机器学习的框架中，如果我们想学习从 x 到 y 的一个映射，这就是一个回归问题，其中 y 是连续变量。这里有监督的意思是指，我们已经知道成对的 x 和 y。某种意思上来说，我们知道了正确答案，剩下的问题就是如何从这些观测值（或者数据集）中抽象出一种映射关系来处理未来的观测（也就是只知道 x 而不知道 y 的情形）。

4.1.2 线性回归模型的核心

前面已经讨论了线性回归的一些基本思想，现在我们需要在统计学和机器学习的术语之间构建一座桥梁，来学习如何构建线性模型。 看下面这个公式：

$$y_i = \alpha + \beta x_i$$

这个等式描述的是变量 $\boldsymbol{x}$ 与变量 $\boldsymbol{y}$ 之间的线性关系。其中，参数 β 控制的是直线的斜率，这里斜率可以理解为变量 $\boldsymbol{x}$ 的单位变化量所对应 $\boldsymbol{y}$ 的变化量。另

① Machine Learning: a Probabilistic Perspective 一书的作者。——译者注

外一个参数 α 可以解释为当 x_i=0 时 y_i 的值，在图中表示，α 就是直线与 y 轴交点的坐标。

计算线性模型参数的方法有很多，最小二乘法是方法之一。每次使用软件去拟合直线的时候，其底层可能就是用的该方法，这种方法返回的 α 和 β 能够让观测到的 $\boldsymbol{y}$ 与预测的 $\boldsymbol{y}$ 之间误差平方的均值最小。这样，估计 α 和 β 就变成了一个最优化问题，最优化问题的目标一般是寻找函数的最小值（或最大值）。最优化并非求解线性模型的唯一方法，同样的问题还可以从概率（贝叶斯）的角度描述。用概率的方式思考带来的优势是：我们在得到最优 α 和 β（与最优化方法求解结果相同）的同时还知道这些参数的不确定性，而最优化方法需要一些其他工作来提供这类信息。此外，我们得到的还有贝叶斯方法的灵活性，这意味着我们可以将模型应用到特定的问题中，比如将正态假设移除，或者构建分层线性模型。

从概率的角度，线性回归模型可以表示成如下形式：

$$\boldsymbol{y} \sim N(\mu = \alpha + \beta \boldsymbol{x}, \sigma = \varepsilon)$$

也就是说，这里假设向量 $\boldsymbol{y}$ 是服从均值为 $\alpha+\beta\boldsymbol{x}$、标准差为 ε 的正态分布。由于我们并不知道 α、β 或者 ε，因此需要对其设置先验，一组合理的先验如下：

$$\alpha \sim N(\mu_\alpha, \sigma_\alpha)$$
$$\beta \sim N(\mu_\beta, \sigma_\beta)$$
$$\varepsilon \sim U(0, h_s)$$

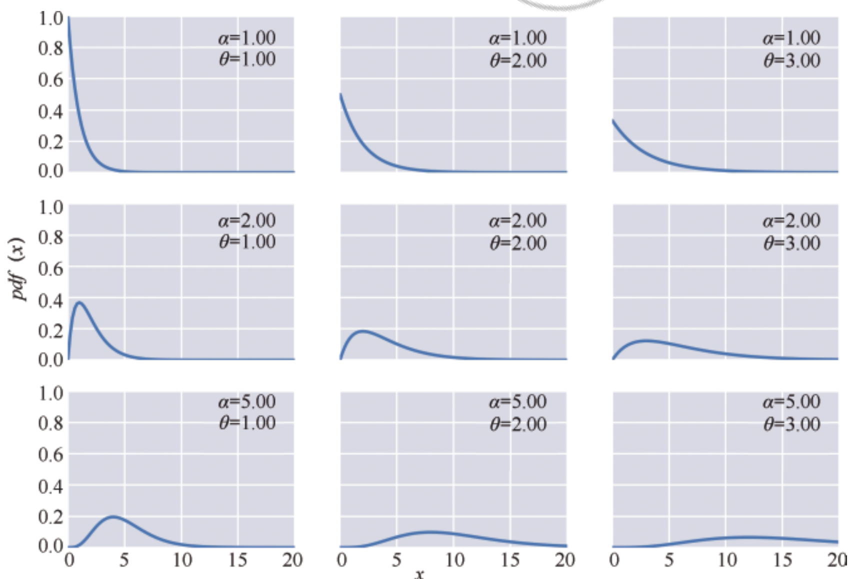
对于 α 的先验，我们可以使用一个分布很平坦的高斯分布，即σ_α相对数据的值域很大。通常我们并不知道截距是多少，其具体的值根据问题不同有很大变化。对于斜率来说同样如此，对很多问题而言，我们至少根据先验知道它是否大于 0。对于 ε 来说，我们可以根据 $\boldsymbol{y}$ 的值域，设置 h_s 为一个比较大的值，例如，将其设为 $\boldsymbol{y}$ 的标准差的 10 倍。在这个贝叶斯模型中，对于单参数线性回归问题，只要有效地选择了相对平坦的先验，我们可以得到和最小二乘方拟合一样的结果。其中均匀分布还可以换成半正态分布或者半柯西分布。通常，半柯西分布作为一个不错的正则先验（具体见第 6 章）很有效。如果我们希望在某些值附近给标准差设置很强的先验，那么可以使用伽马分布。在许多软件库中，伽马分布的

默认参数看起来有点奇怪，不过幸运地是 PyMC3 可以让我们同时使用形状和比例或者均值和方差来定义。

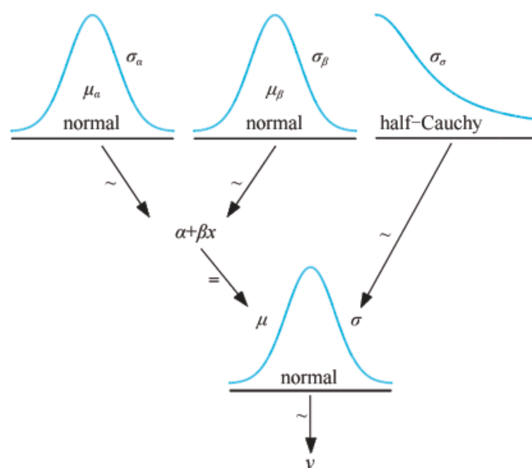
继续讨论线性回归之前，先看一下伽马分布在不同参数下的形状：

```
rates = [1, 2, 5]
scales = [1, 2, 3]

x = np.linspace(0, 20, 100)
f, ax = plt.subplots(len(rates), len(scales), sharex=True, sharey=True)
for i in range(len(ass)):
    for j in range(len(scales)):
        rate = rates[i]
        scale = scales[j]
        rv = stats.gamma(a=rate, scale=scale)
        ax[i,j].plot(x, rv.pdf(x))
        ax[i,j].plot(0, 0,
        label="$\\alpha$ = {:3.2f}\n$\\theta$ = {:3.2f}".format(rate,
scale), alpha=0)
        ax[i,j].legend()
ax[2,1].set_xlabel('$x$')
ax[1,0].set_ylabel('$pdf(x)$')
```

再来看线性回归模型，借助 Kruschke 图，我们有下面这张图：

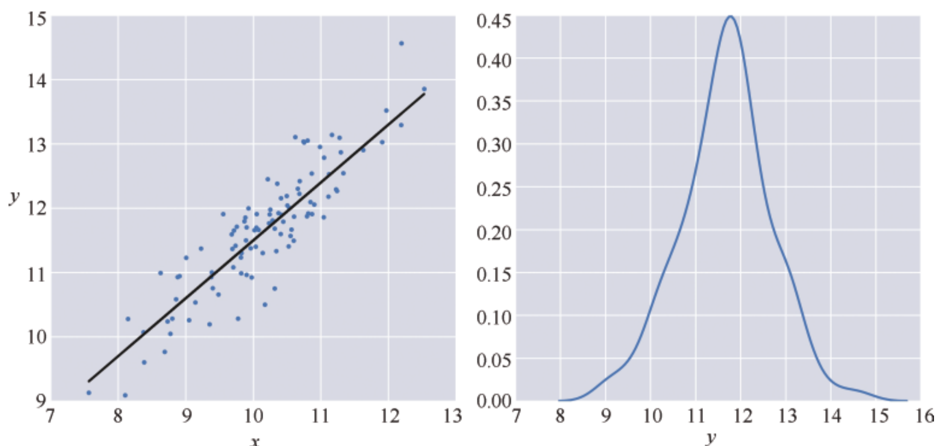


现在需要将数据输出给模型，这里再次使用了合成数据，来构建对模型的直观认识。先假设已经知道参数的真实值，然后构造出数据集，最后再利用模型将其找出来。

```
np.random.seed(314)
N = 100
alfa_real = 2.5
beta_real = 0.9
eps_real = np.random.normal(0, 0.5, size=N)

x = np.random.normal(10, 1, N)
y_real = alfa_real + beta_real * x
y = y_real + eps_real

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.plot(x, y, 'b.')
plt.xlabel('$x$', fontsize=16)
plt.ylabel('$y$', fontsize=16, rotation=0)
plt.plot(x, y_real, 'k')
plt.subplot(1,2,2)
sns.kdeplot(y)
plt.xlabel('$y$', fontsize=16)
```



现在用 PyMC3 来拟合数据，代码看起来和前面的模型都差不多，不过留意这里 μ 是通过 `pm.Deterministic` 来定义的，也就是说该变量是确定的。目前为止我们见过的所有变量都是随机的，也就是说，每次求随机变量值的时候，都会得到一个不同的值。而在这里，确定的变量意味着其值完全由参数决定，尽管在下面的代码中这些参数是随机的。如果我们声明一个变量是确定的，那么 PyMC3 会将其保存在迹中。


```
with pm.Model() as model:
    alpha = pm.Normal('alpha', mu=0, sd=10)
    beta = pm.Normal('beta', mu=0, sd=1)
    epsilon = pm.HalfCauchy('epsilon', 5)

    mu = pm.Deterministic('mu', alpha + beta * x)
    y_pred = pm.Normal('y_pred', mu=mu, sd=epsilon, observed=y)

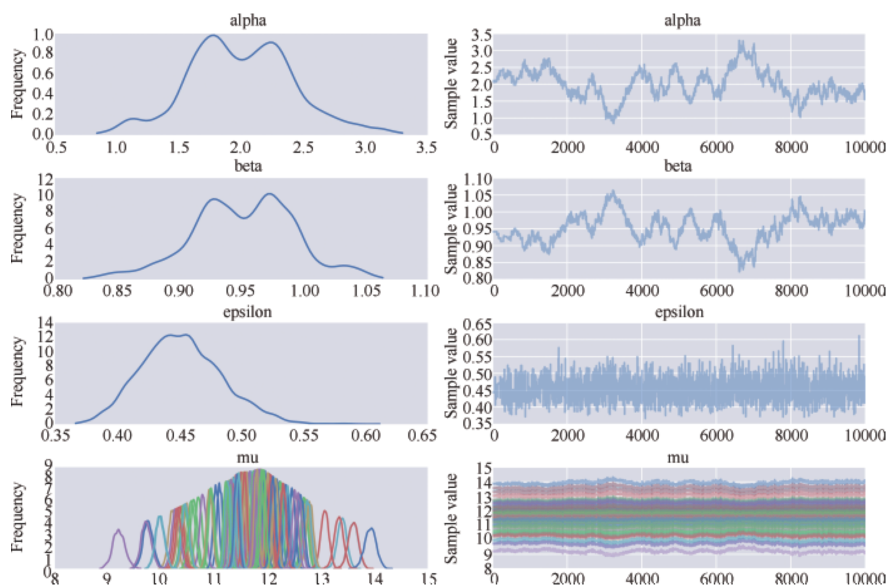
    start = pm.find_MAP()
    step = pm.Metropolis()
    trace = pm.sample(10000, step, start)
```

此外，还可以省略固定变量的声明而直接写为如下语句：

```
y_pred = pm.Normal('y_pred', mu= alpha + beta * x, sd=epsilon,
observed=y)
```

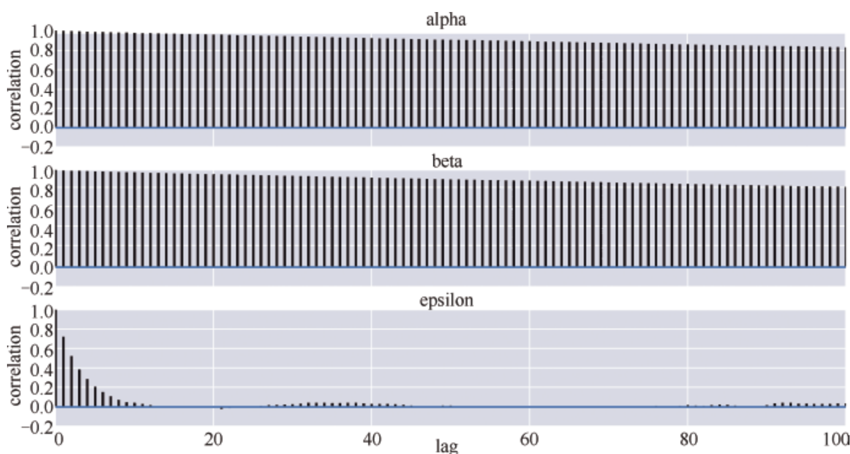
`traceplot` 函数返回的图像很有趣。注意其中 α 和 β 参数在缓慢地上下波动；对比之下，只有 ε 的迹呈现了较好的混合度。显然，出问题了。

```
pm.traceplot(chain);
```



上面的自相关图表明，α 和 β 有很强的自相关性，而 ε 没有。接下来我们会看到，这是线性模型的典型表现。注意这里通过给 autocorrplot 传递 varnames 参数，没有画出确定的变量 μ。

```
varnames = ['alpha', 'beta', 'epsilon']
pm.autocorrplot(trace, varnames)
```

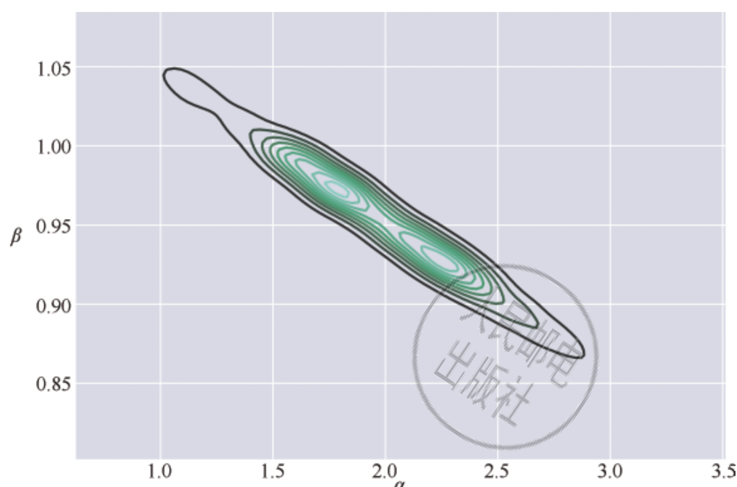


4.1.3　线性模型与高自相关性

前面的模型中，α 和 β 有很糟糕的自相关性，这意味着采样很差，而且相比

实际的采样数，有效的采样很少。为什么呢？其实我们是被自己的假设误导了。事实上，不论我们用哪条直线去拟合数据，它们都会穿过一点，x 的均值和 y 的均值点。因此，拟合直线的过程相当于将直线固定在数据的中心上进行旋转，斜率越大截距越小。根据模型的定义，两个参数是相关的，如果将后验画出来的话可以很清楚地看到这点（这里暂时忽略 ε）。

```
sns.kdeplot(trace['alpha'], trace['beta'])
plt.xlabel(r'$\alpha$', fontsize=16)
plt.ylabel(r'$\beta$', fontsize=16, rotation=0)
```



可以看到，后验（除了 ε 之外）呈斜对角形状，对于类似 Metropolis-Hastings 的算法，这可能会存在问题。因为如果给每个独立的参数设置一个较大的步长，那么很可能会落在高概率区域以外；只有步长设得很小的时候，被接受的概率才会较高。不管哪种方式我们都会得到很高的自相关性和较差的混合度，而且数据的维度越高，这种情况越严重，因为总的参数空间要比可能的参数空间增长得快得多，关于这点可以查阅维基百科中维数灾难的内容进行了解。

在继续深入之前，请允许我澄清一点，前面提到的拟合直线会穿过数据的均值点只在最小二乘方算法的假设下成立。使用贝叶斯方法之后，这个限制被放松了。后面的例子中我们可以看到，通常直线会在 x 和 y 的均值附近而不是正好穿过均值。不过没关系，自相关性与直线固定在某一点附近的假设仍然是成立的，关于高自相关性问题我们需要了解的就是这么多了，接下来我们会从两个方面理

解和解决高自相关性问题。

运行之前先修改数据

解决问题的一个简单办法是先将 $\boldsymbol{x}$ 中心化，也就是说，对于每个点 x_i，减去 $\boldsymbol{x}$ 的均值（$\overline{x}$）。

$$\boldsymbol{x}' = \boldsymbol{x} - \overline{\boldsymbol{x}}$$

这样做的结果是 $\boldsymbol{x}'$ 的中心在 0 附近，从而修改斜率时旋转点变成了截距点，参数空间也会变得不那么自相关。

中心化不仅仅是一种计算技巧，同时有利于解释数据。截距是指当 x_i=0 时 y_i 的值，不过对于许多问题而言，截距并没有什么实际的意义。例如，对于身高或者体重这类数值，当值为 0 时，并没有实际的意义，因而截距对于理解数据也就没有任何帮助，对于另外一些问题，估计出截距可能很有用，因为在实验中我们可能无法测量出 x_i=0 的情况，但截距的估计值可以为我们提供有价值的信息。不管怎么说，推断都有其局限性，应当谨慎使用。

根据问题和受众不同，我们可能需要汇报中心化之前或者之后估计到的参数值。如果我们需要汇报的是中心化之前的参数，那么可以像下面这样将参数转换成原来的尺度：

$$\alpha = \alpha' - \beta' \overline{\boldsymbol{x}}$$

上面的公式可以通过以下公式推导出来：

$$\boldsymbol{x}' = \boldsymbol{x} - \overline{\boldsymbol{x}}$$
$$\boldsymbol{y} = \alpha' + \beta' \boldsymbol{x}' + \varepsilon$$
$$\boldsymbol{y} = \alpha' + \beta'(\boldsymbol{x} - \overline{\boldsymbol{x}}) + \varepsilon$$
$$\boldsymbol{y} = \alpha' - \beta' \overline{\boldsymbol{x}} + \beta' \boldsymbol{x} + \varepsilon$$

然后可以得出：

$$\alpha = \alpha' + \beta' \overline{\boldsymbol{x}}$$

$$\beta = \beta'$$

更进一步，在运行模型之前，我们可以对数据进行标准化处理。标准化在

统计学和机器学习中是一种常见的数据处理手段，这是因为许多算法对于标准化之后的数据效果更好。标准化的过程是在中心化之后再除以标准差，其数学形式如下：

$$x' = \frac{x - \bar{x}}{x_{sd}}$$

$$y' = \frac{y - \bar{y}}{y_{sd}}$$

标准化的好处之一是我们可以对数据使用相同的弱先验，而不必关心数据的具体值域有多大，因为我们已经对数据进行了尺度变换。对于标准化之后的数据，截距通常在 0 附近，斜率在 -1 ～ 1 附近。标准化之后的数据可以使用**标准分数**（Z-score）来描述参数。如果某人声称一个参数的标准分数值为 1.3，那么我们就知道该值在标准化之前位于均值附近 1.3 倍的标准差处。标准分数每变化一个单位，那么对应原始数据中变化 1 倍的标准差。这一点在分析多个变量时很有用，因为所有的参数都在同一个尺度，从而简化了对数据的解释。

更换采样方法

另外一种解决高自相关性的办法是使用不同的采样方法。NUTS 算法与 Metropolis 算法相比，在类似受限的对角空间中遇到的困难小一些。原因是 NUTS 是根据后验的曲率来移动的，因而更容易沿着对角空间移动。NUTS 算法每走一步要比 Metropolis 算法更慢，不过要得到一个合理的后验近似值所需要的步数更少。

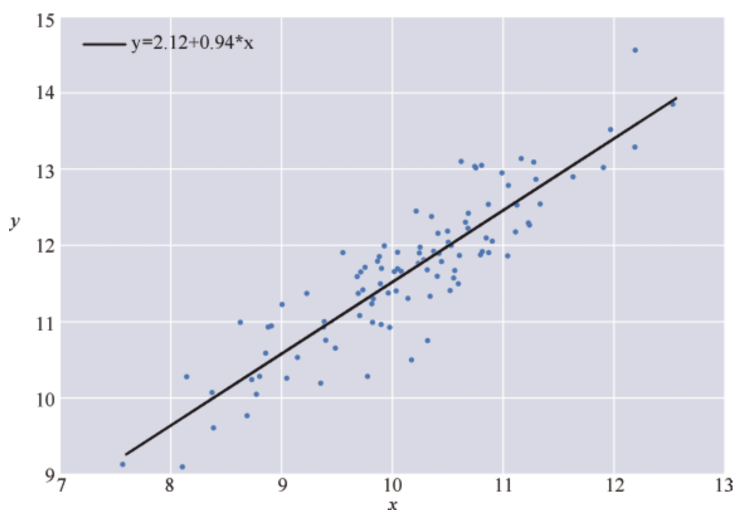
下面将讨论使用 NUTS 采样方法得到的结果。

4.1.4　对后验进行解释和可视化

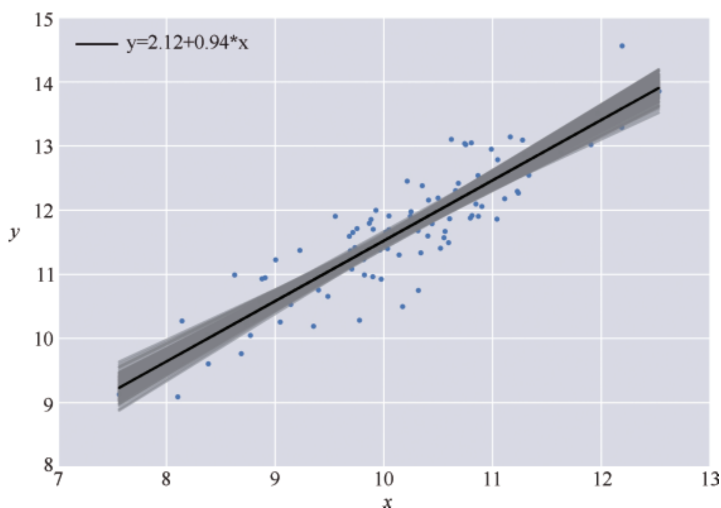
前面我们已经知道了如何使用 PyMC3 中的 `traceplot` 和 `df_summary` 函数或者一些自定义的函数探索后验。对于线性回归，一种更好的表现方式是将拟合数据的平均直线与 α 和 β 的均值同时表示在图上：

```
plt.plot(x, y, 'b.');
alpha_m = trace_n['alpha'].mean()
beta_m = trace_n['beta'].mean()
plt.plot(x, alpha_m + beta_m * x, c='k', label='y = {:.2f} + {:.2f} * 
x'.format(alpha_m, beta_m))
```

```
plt.xlabel('$x$', fontsize=16)
plt.ylabel('$y$', fontsize=16, rotation=0)
plt.legend(loc=2, fontsize=14)
```



或者，我们还可以从后验采样中画出半透明的直线来表示后验的不确定性。代码如下。




```
plt.plot(x, y, 'b.');

idx = range(0, len(trace_n['alpha']), 10)
plt.plot(x, trace_n['alpha'][idx] + trace_n['beta'][idx] *x[:,np.
newaxis], c='gray', alpha=0.5);
```

```
plt.plot(x, alpha_m + beta_m * x, c='k', label='y = {:.2f} + {:.2f} *
x'.format(alpha_m, beta_m))

plt.xlabel('$x$', fontsize=16)
plt.ylabel('$y$', fontsize=16, rotation=0)
plt.legend(loc=2, fontsize=14)
```

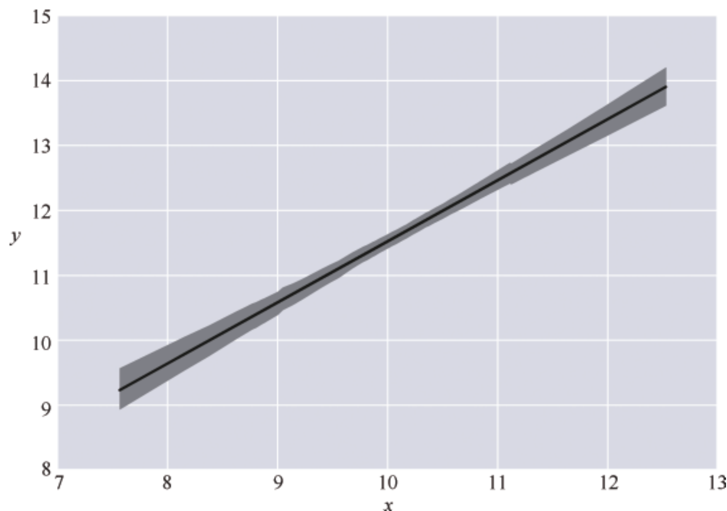
可以看到，上图中间部分的不确定性较低，不过直线并没有都相交于一个点（后验并不强制所有的直线都穿过均值点）。

半透明的直线看起来不错，不过我们可能想给这个图增加点更酷的东西：用半透明的区间描述 μ 的最大后验密度（HPD）区间。注意这也是在模型中将变量 μ 定义成一个确定值的主要原因，简化以下代码：

```
plt.plot(x, alpha_m + beta_m * x, c='k', label='y = {:.2f} + {:.2f} *
x'.format(alpha_m,beta_m))

idx = np.argsort(x)
x_ord = x[idx]
sig = pm.hpd(trace_n['mu'], alpha=.02)[idx]
plt.fill_between(x_ord, sig[:,0], sig[:,1], color='gray')

plt.xlabel('$x$', fontsize=16)
plt.ylabel('$y$', fontsize=16, rotation=0)
```

另外一种方式是画预测值 $\hat{y}$ 的 HPD（例如 95% 和 50%）区间。也就是说，我们想要根据模型看到未来 95% 和 50% 的数据的分布范围。我们在图中将

50%HPD 区间用深灰色区域表示，将 95%HPD 区间用浅灰色表示。利用 PyMC3 中的 sample_ppc 函数可以很容易得到预测值的采样值。

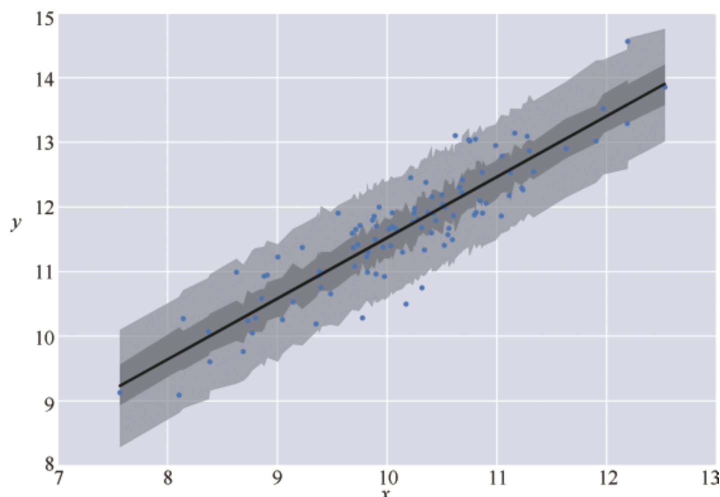
```
ppc = pm.sample_ppc(chain_n, samples=1000, model=model_n)
```

然后我们可以画出结果。

```
plt.plot(x, y, 'b.')
plt.plot(x, alpha_m + beta_m * x, c='k', label='y = {:.2f} + {:.2f} *
x'.format(alpha_m, beta_m))

sig0 = pm.hpd(ppc['y_pred'], alpha=0.5)[idx]
sig1 = pm.hpd(ppc['y_pred'], alpha=0.05)[idx]
plt.fill_between(x_ord, sig0[:,0], sig0[:,1], color='gray', alpha=1)
plt.fill_between(x_ord, sig1[:,0], sig1[:,1], color='gray', alpha=0.5)

plt.xlabel('$x$', fontsize=16)
plt.ylabel('$y$', fontsize=16, rotation=0)
```



图中 HPD 区间的边界不是很规则，原因是：为了画出这个图，我们是从观测值 x 而不是从连续区间中得到的后验预测采样，此外，fill_between 函数只是简单地对相邻点之间进行线性差值。留意锯齿的尖锐程度与数据数量之间的关系，可以通过从 y_pred 中获取更多的采样值来减少不规则的程度。

4.1.5 皮尔逊相关系数

许多时候，我们希望衡量两个变量之间的（线性）依赖关系。衡量两个变量

之间线性相关性最常见的指标是皮尔逊相关系数，通常用小写的 r 表示。如果 r 的值为 +1，我们称这两个变量完全正相关，也就是说一个变量随着另外一个变量的增加而增加；如果 r 的值为 -1，那么称为完全负相关，也就是说一个变量随着另一个变量的增加而减少；当 r 为 0 时，我们称两个变量之间没有线性相关性。皮尔逊相关系数并不涉及非线性相关性。人们很容易将皮尔逊相关系数与回归中的斜率弄混淆，查看以下链接中的图就可以明白二者本质上是两个不同的量：https://en.wikipedia.org/wiki/Correlation_and_dependence#/media/File:Correlation_examples2.svg

下面的公式可以在某种程度上减轻你的疑惑：

$$r = \beta \frac{\sigma(\boldsymbol{x})}{\sigma(\boldsymbol{y})}$$

也就是说，只有在 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 的标准差相等时，皮尔逊相关系数才与斜率相等。当我们对数据标准化时，上式是成立的。需要注意：

- 皮尔逊相关系数衡量的是两个变量之间的相关性程度，其值位于 [−1,1] 区间内，与数据的尺度无关；
- 斜率表示的是 $\boldsymbol{x}$ 变化一个单位时，$\boldsymbol{y}$ 的变化量，可以取任意实数。

皮尔逊相关系数与决定系数（Coefficient of Determination）之间有联系，对于线性回归模型而言，决定系数就是皮尔逊相关系数的平方，即 r^2（或者 R^2）。决定系数用于度量变量的变异中可以用自变量解释部分所占的比例。

现在，我们扩展下线性回归模型，利用 PyMC3 从两个方面计算 r 和 r^2。

- 一种方式是使用前面看到的皮尔逊相关系数与斜率之间关系的公式，将其记作变量 `rb`。
- 另一种方式与最小二乘方算法相关，这里暂时跳过其来源细节，将其记作变量 `rss`。仔细阅读代码可以看到，变量 `ss_reg` 衡量的是拟合的直线与数据均值之间的离差，与模型中的方差成比例，其公式与方差很像，不过没有除以数据的个数。变量 `ss_tot` 与预测值的方差成比例。

完整的模型如下：

```
with pm.Model() as model_n:
    alpha = pm.Normal('alpha', mu=0, sd=10)
```

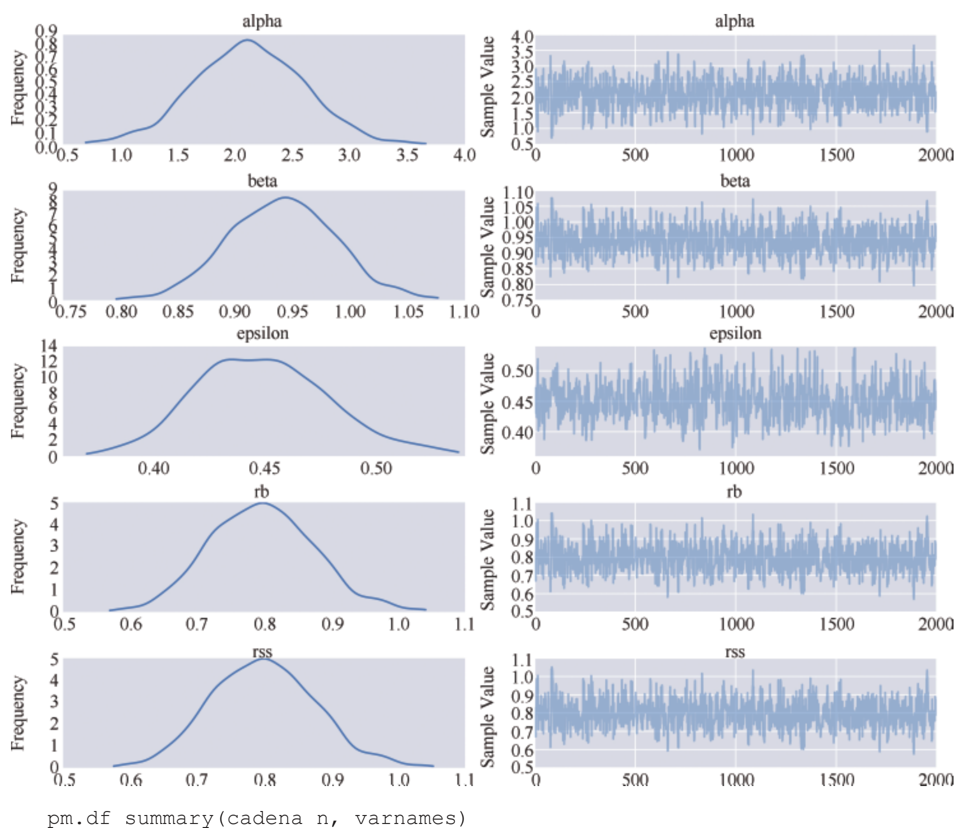
```
    beta = pm.Normal('beta', mu=0, sd=1)
    epsilon = pm.HalfCauchy('epsilon', 5)

    mu = alpha + beta * x
    y_pred = pm.Normal('y_pred', mu=mu, sd=epsilon, observed=y)

    rb = pm.Deterministic('rb', (beta * x.std() / y.std()) ** 2)

    y_mean = y.mean()
    ss_reg = pm.math.sum((mu - y_mean) ** 2)
    ss_tot = pm.math.sum((y - y_mean) ** 2)
    rss = pm.Deterministic('rss', ss_reg/ss_tot)

    start = pm.find_MAP()
    step = pm.NUTS()
    trace_n = pm.sample(2000, step=step, start=start)
pm.traceplot(chain_n)
```




```
pm.df_summary(cadena_n, varnames)
```

	mean	sd	mc_error	hpd_2.5	hpd_97.5
alpha	2.11	0.49	1.87e-02	1.21	3.13
beta	0.94	0.05	1.82e-03	0.84	1.03
epsilon	0.45	0.03	1.30e-03	0.39	0.52
rb	0.80	0.08	3.09e-03	0.64	0.96
rss	0.80	0.08	3.22e-03	0.64	0.95

根据多元高斯分布计算皮尔逊相关系数

另一种计算皮尔逊相关系数的方法是估计一个多元高斯分布的方差矩阵。我们暂时只考虑二维的情况，一旦理解了两个变量的情况之后，理解更高维度就会很容易了。为了充分描述二元高斯分布，我们需要两个均值（或者一个长度为 2 的向量），每个均值对应一个边缘高斯分布，此外还需要两个标准差，这么说其实不太准确，事实上我们需要一个像下面这样的 2×2 的**协方差矩阵**：

$$\sum = \begin{bmatrix} \sigma_{x1}^2 & \rho\sigma_{x2}\sigma_{x1} \\ \rho\sigma_{x2}\sigma_{x1} & \sigma_{x2}^2 \end{bmatrix}$$

其中，$\sum$是大写的希腊字母希格玛，通常用其表示协方差矩阵。在主对角线上的两个元素分别是每个变量的方差，用标准差σ_{x1}和σ_{x1}的平方表示。剩余的两个元素分别是协方差（变量之间的共变），用每个变量标准差和ρ（变量之间的皮尔逊相关系数）来表示。注意这里只有一个ρ，原因是我们只有两个变量，如果有 3 个变量的话，对应的会有 3 个ρ，如果有 4 个变量的话，对应会有 6 个ρ。计算这些数需要用到二项系数，回忆一下第 1 章中的二项分布。

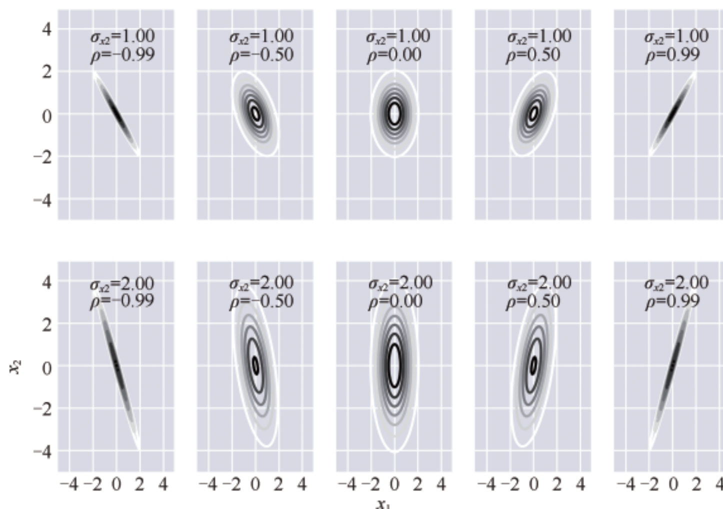
下面的代码生成了一些二维高斯分布的等值线图，其中均值都固定在（0,0），标准差σ_{x1}取 1，另外一个标准差σ_{x2}分别取 1 或者 2，皮尔逊相关系数取 -1 到 1 之间的不同值。

```
sigma_x1 = 1
sigmas_x2 = [1, 2]
rhos = [-0.99, -0.5, 0, 0.5, 0.99]

x, y = np.mgrid[-5:5:.1, -5:5:.1]
pos = np.empty(x.shape + (2,))
pos[:, :, 0] = x; pos[:, :, 1] = y
```

```
f, ax = plt.subplots(len(sigmas_x2), len(rhos), sharex=True, sharey=True)

for i in range(2):
    for j in range(5):
        sigma_x2 = sigmas_x2[i]
        rho = rhos[j]
        cov = [[sigma_x1**2, sigma_x1*sigma_x2*rho],
          [sigma_x1*sigma_x2*rho, sigma_x2**2]]
        rv = stats.multivariate_normal([0, 0], cov)
        ax[i,j].contour(x, y, rv.pdf(pos))
        ax[i,j].plot(0, 0,
        label="$\\sigma_{{x2}}$ = {:3.2f}\n$\\rho$ = {:3.2f}".format(sigma_
x2, rho), alpha=0)
        ax[i,j].legend()
ax[1,2].set_xlabel('$x_1$')
ax[1,0].set_ylabel('$x_2$')
```



了解了多维高斯分布之后，我们就可以拿它来估计皮尔逊相关系数了。由于我们并不知道协方差矩阵，可以先为其设置一个先验。一种做法是使用威沙特分布（Wishart distribution）[①]，威沙特分布是多维正态分布逆协方差矩阵的共轭先验，可以看作是前面见过的伽马分布在高维空间的一般形式，也可以看作卡方分布（chi squared distribution）的一般形式。另一种做法是使用 LKJ 先验，该先验用于相关性矩阵的（不是协方差矩阵），如果考虑相关性的话，使用起来更方便

① Wishart distribution 虽然是协方差矩阵的共轭先验，在 MCMC 中它的混合度很差而且很难从中准确采样，我们推荐使用 LKJCorr 或 LKJCholeskyCov 作为协方差矩阵的先验。——译者注

一些。这里我们讨论第 3 种做法，直接为σ_{x1}、σ_{x2}和ρ设置先验，然后用这些值手动构造协方差矩阵。

```
with pm.Model() as pearson_model:

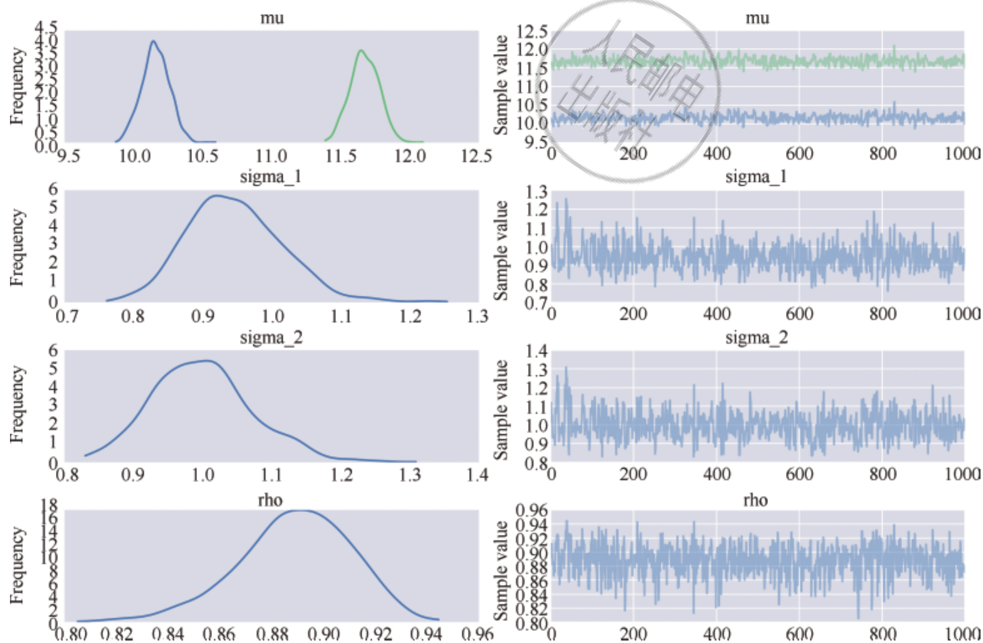
    mu = pm.Normal('mu', mu=data.mean(0), sd=10, shape=2)

    sigma_1 = pm.HalfNormal('sigma_1', 10)
    sigma_2 = pm.HalfNormal('sigma_2', 10)
    rho = pm.Uniform('rho', -1, 1)

    cov = pm.math.stack(([sigma_1**2, sigma_1*sigma_2*rho], [sigma_1*
sigma_2*rho, sigma_2**2]))

    y_pred = pm.MvNormal('y_pred', mu=mu, cov=cov, observed=data)

    start = pm.find_MAP()
    step = pm.NUTS(scaling=start)
    trace_p = pm.sample(1000, step=step, start=start)
```



注意这里得到的ρ是皮尔逊相关系数，而前面的例子中得到的是皮尔逊相关系数的平方，考虑这点之后你会发现这里得到的结果与前面例子中的结果是一致的。

4.2 鲁棒线性回归

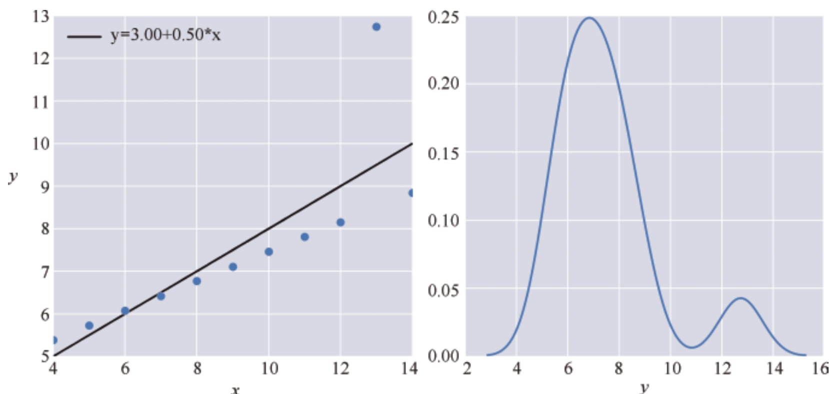
在许多情况下，假设数据服从高斯分布是非常合理的。我们假设数据符合高斯特性，并不是说数据真的就是符合高斯分布的，而是说我们认为高斯分布对于我们的问题而言是一个合理的近似。前面我们知道了，有时候高斯假设并不成立，例如出现异常值的时候，利用 t 分布，可以有效地解决异常值的问题，从而得到更鲁棒的推断。类似的思想同样可以应用到线性回归问题中。

为了验证 t 分布确实能增加线性回归的鲁棒性，这里我们使用一个非常简单的数据集：安斯库姆四重奏（Anscombe's quartet）中的第 3 组数据。如果你不知道安斯库姆四重奏数据集，可以在维基百科上查看，这里我们从 seaborn 中加载它。

```
ans = sns.load_dataset('anscombe')
x_3 = ans[ans.dataset == 'III']['x'].values
y_3 = ans[ans.dataset == 'III']['y'].values
```

然后来看看这个数据集长什么样。

```
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
beta_c, alpha_c = stats.linregress(x_3, y_3)[:2]
plt.plot(x_3, (alpha_c + beta_c* x_3), 'k', label='y ={:.2f} + {:.2f} *
x'.format(alpha_c, beta_c))
plt.plot(x_3, y_3, 'bo')
plt.xlabel('$x$', fontsize=16)
plt.ylabel('$y$', rotation=0, fontsize=16)
plt.legend(loc=0, fontsize=14)
plt.subplot(1,2,2)
sns.kdeplot(y_3);
plt.xlabel('$y$', fontsize=16)
```



现在我们用 t 分布替换模型中的高斯分布，这个改变需要引入正态参数 v，有关该参数的含义，可以参照前一章的内容。

在下面的模型中，我们使用了移位指数分布来避免 nu 的值接近 0，因为非移位指数分布对于 0 附近的值赋予了太高的权重。根据我的经验，对于没有异常点或者含有少量异常点的数据集而言，使用非移位指数分布就够了，不过对于某些包含极限异常值的数据（或者是只有少量聚集点的数据集）而言，例如我们用到的安斯库姆四重奏数据集的第 3 组，移位指数分布更合适。这些建议都只是基于我（或者别人）处理某些数据集或者问题的经验。此外，正态参数 v 的一些常见先验还有 gamma(2,0.1) 或者 gamma(mu=20, sd=15)。

注意，在 PyMC3 中以下划线 _ 结尾的变量，如 nu_（译者注：新版本的 PyMC3 中写成 _nu_），对于用户是不可见的。

```
with pm.Model() as model_t:
    alpha = pm.Normal('alpha', mu=0, sd=100)
    beta = pm.Normal('beta', mu=0, sd=1)
    epsilon = pm.HalfCauchy('epsilon', 5)
    nu = pm.Deterministic('nu', pm.Exponential('nu_', 1/29) + 1)

    y_pred = pm.StudentT('y_pred', mu=alpha + beta * x_3, sd=epsilon, nu=nu, observed=y_3)

    start = pm.find_MAP()
    step = pm.NUTS(scaling=start)
    trace_t = pm.sample(2000, step=step, start=start)
```

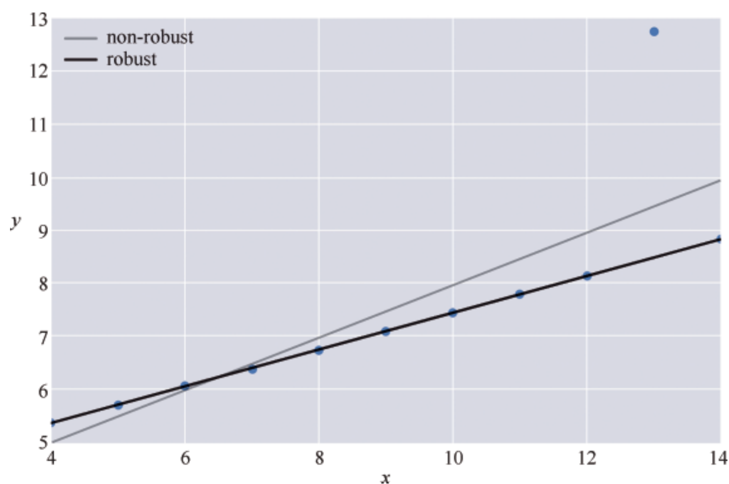
这里为了节省版面，省略一些代码和图（如迹和自相关性的图），不过你自己尝试的时候可别这么做。我在这里只画出了平均的拟合直线，此外还用 SciPy 中的 linregress 函数画了一条非鲁棒的直线。你可能需要尝试根据前面例子中的贝叶斯方法画出对应的直线。

```
beta_c, alpha_c = stats.linregress(x_3, y_3)[:2]

plt.plot(x_3, (alpha_c + beta_c * x_3), 'k', label='non-robust', alpha=0.5)
plt.plot(x_3, y_3, 'bo')
alpha_m = trace_t['alpha'].mean()
beta_m = trace_t['beta'].mean()
plt.plot(x_3, alpha_m + beta_m * x_3, c='k', label='robust')

plt.xlabel('$x$', fontsize=16)
```

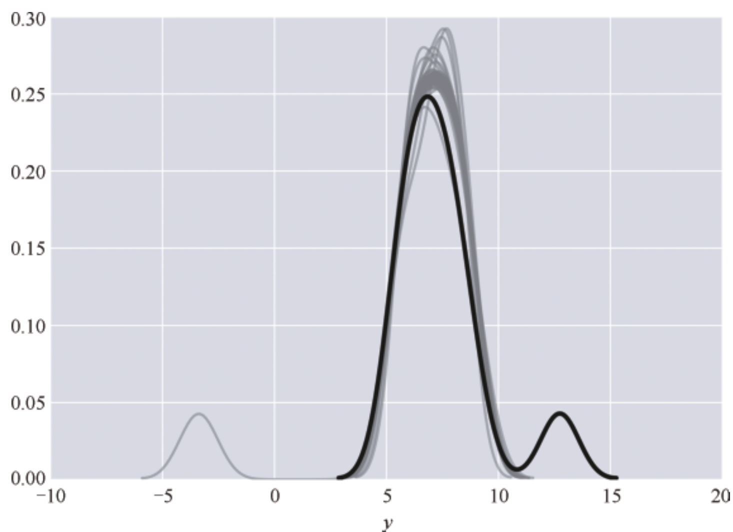
```
plt.ylabel('$y$', rotation=0, fontsize=16)
plt.legend(loc=2, fontsize=12)
```



执行下后验预测检查，看看模型是如何捕捉数据的。这里可以用 PyMC3 替我们从后验中采样：

```
ppc = pm.sample_ppc(chain_t, samples=200, model=model_t, random_seed=2)
for y_tilde in ppc['y_pred']:
    sns.kdeplot(y_tilde, alpha=0.5, c='g')
sns.kdeplot(y_3, linewidth=3)
```

然后会看到类似下面的图：



对大多数数据，我们都拟合得相当不错，而且注意，这里不仅对中心部分的数据拟合得很好，对于异常点部分也拟合得不错。对于当前的目标来说，这个模型表现得不错，不需要更多的改进了。不过，在某些问题中，我们可能不希望有负数，这种情况下，我们可能需要回过头修改下模型，限制 y 为正数。

4.3　分层线性回归

前面，我们学习了分层模型的基础知识，现在我们可以将这些概念应用到线性回归，在组这一层同时对多个组建模并进行估计。和前面一样，这里是通过引入超先验实现的。 我们先创建 8 个相关的数据组，其中包括一个只有一个点的组。

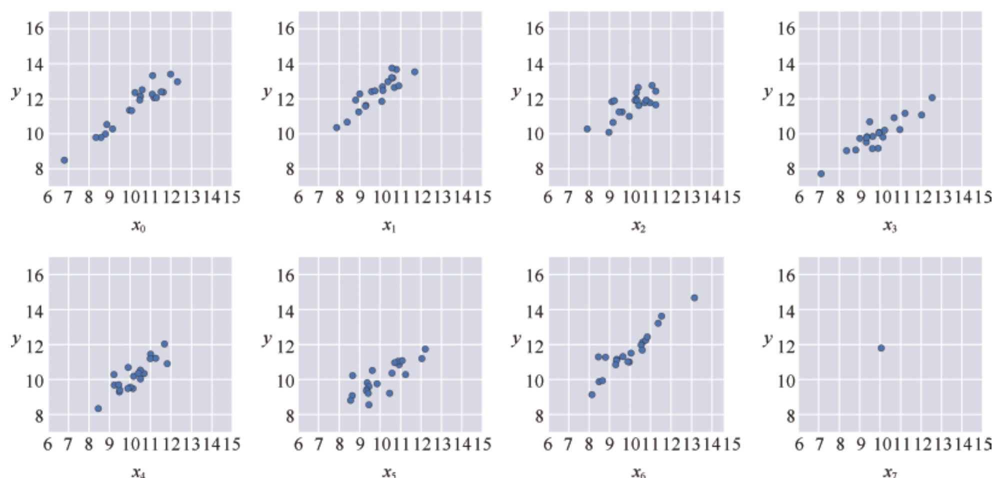
```
N = 20
M = 8
idx = np.repeat(range(M-1), N)
idx = np.append(idx, 7)

alpha_real = np.random.normal(2.5, 0.5, size=M)
beta_real = np.random.beta(60, 10, size=M)
eps_real = np.random.normal(0, 0.5, size=len(idx))

y_m = np.zeros(len(idx))
x_m = np.random.normal(10, 1, len(idx))
y_m = alpha_real[idx] + beta_real[idx] * x_m  + eps_real
```

我们的数据如下：

```
j, k = 0, N
for i in range(M):
    plt.subplot(2,4,i+1)
    plt.scatter(x_m[j:k], y_m[j:k])
    plt.xlim(6, 15)
    plt.ylim(7, 17)
    j += N
    k += N
plt.tight_layout()
```



在输入到模型之前，先对其进行中心化处理：

```
x_centered = x_m - x_m.mean()
```

首先，和前面的做法一样，先用非多层的模型拟合，唯一的区别是需要增加部分代码将 α 转换到原始的尺度。

```
with pm.Model() as unpooled_model:
    alpha_tmp = pm.Normal('alpha_tmp', mu=0, sd=10, shape=M)
    beta = pm.Normal('beta', mu=0, sd=10, shape=M)
   epsilon = pm.HalfCauchy('epsilon', 5)

    nu = pm.Exponential('nu', 1/30)

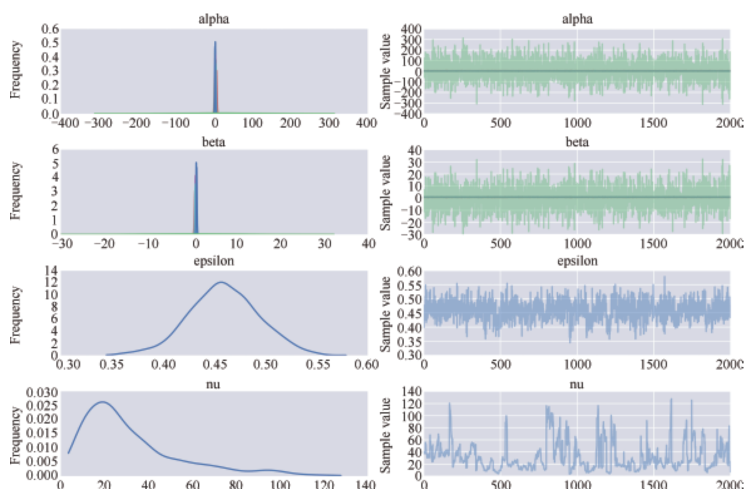
    y_pred = pm.StudentT('y_pred', mu=alpha_tmp[idx] + beta[idx]* x_centered,
sd=epsilon, nu=nu, observed=y_m)

    alpha = pm.Deterministic('alpha', alpha_tmp - beta *x_m.mean())

    start = pm.find_MAP()
    step = pm.NUTS(scaling=start)
    trace_up = pm.sample(2000, step=step, start=start)
```

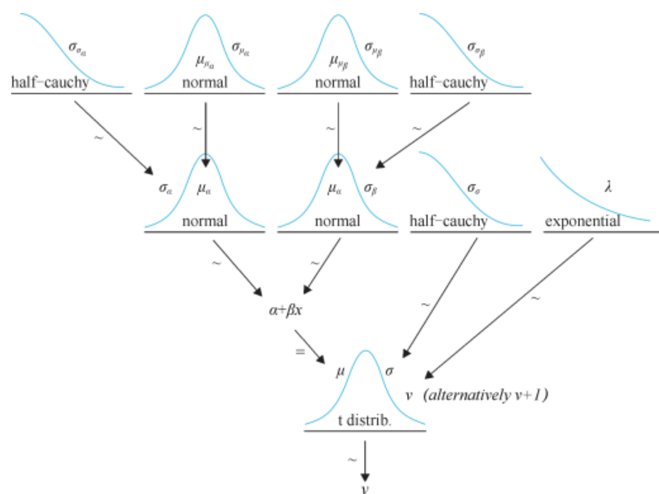
从结果中可以看到，除了其中一组 α 和 β 参数，大多数情况下结果都很正常。根据它们的迹来看，似乎这一组参数一直在自由移动而没有收敛。

```
varnames=['alpha', 'beta', 'epsilon', 'nu']
pm.traceplot(trace_up, varnames);
```



显然，用一条唯一的直线去拟合一个点是不合适的，我们至少需要两个点，或者参数 α 和 β 是有界的，除非我们能提供一些额外的信息（比如加入先验），给 α 加入一个很强的先验能够得到一组明确定义的先验，即使我们的数据中只有一个点。另一种方式是通过构建多层模型往模型中加入信息，这主要是因为多层模型中组与组之间的信息能够共享，这一点对于已经有不同分组的稀疏数据非常有用。这里我们用到的例子中将数据稀疏性推向了极致（其中一组只有一个数据），目的是将问题描述得更清楚一些。

现在我们实现一个与前面线性回归模型相同的多层模型，不过这次用的是超先验，你可以从下面的 Kruschke 图中看到。



用 PyMC3 代码实现的模型与之前模型的主要区别如下。

- 增加了超先验。
- 增加了几行代码将参数转换到中心化之前的尺度。记住这并非强制的，我们完全可以将参数保留在转换后的尺度上，只是对结果进行解释的时候需要小心。
- 使用了 ADVI 而不是 `find_MAP()` 函数去初始化，为 NUTS 算法提供了一个协方差缩放矩阵。如果你不记得什么是 ADVI 了，可以看一下第 2 章中的相关理论描述以及第 2 章练习部分更偏实际一些的讨论。

```
with pm.Model() as hierarchical_model:
    alpha_tmp_mu = pm.Normal('alpha_tmp_mu', mu=0, sd=10)
    alpha_tmp_sd = pm.HalfNormal('alpha_tmp_sd', 10)
    beta_mu = pm.Normal('beta_mu', mu=0, sd=10)
    beta_sd = pm.HalfNormal('beta_sd', sd=10)


    alpha_tmp = pm.Normal('alpha_tmp', mu=alpha_tmp_mu, sd=alpha_tmp_
sd, shape=M)
    beta = pm.Normal('beta', mu=beta_mu, sd=beta_sd, shape=M)
    epsilon = pm.HalfCauchy('epsilon', 5)
    nu = pm.Exponential('nu', 1/30)

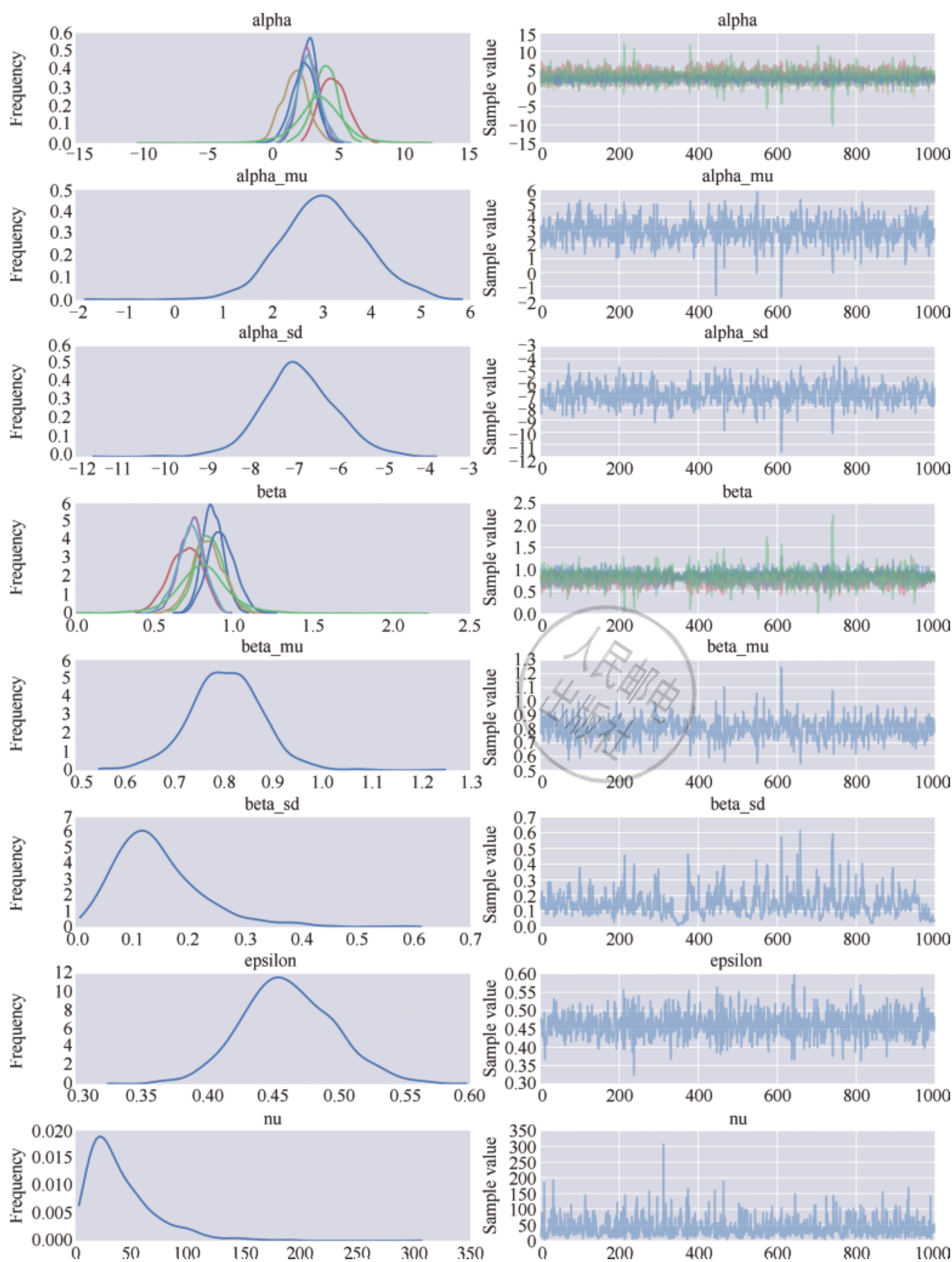
    y_pred = pm.StudentT('y_pred', mu=alpha_tmp[idx] + beta[idx] * x_
centered, sd=epsilon, nu=nu, observed=y_m)

    alpha = pm.Deterministic('alpha', alpha_tmp - beta * x_m.mean())
    alpha_mu = pm.Deterministic('alpha_mu', alpha_tmp_mu - beta_mu * x_
m.mean())
    alpha_sd = pm.Deterministic('alpha_sd', alpha_tmp_sd - beta_mu * x_
m.mean())

    mu, sds, elbo = pm.variational.advi(n=100000, verbose=False)
    cov_scal = np.power(hierarchical_model.dict_to_array(sds), 2)
    step = pm.NUTS(scaling=cov_scal, is_cov=True)
    trace_hm = pm.sample(1000, step=step, start=mu)
```

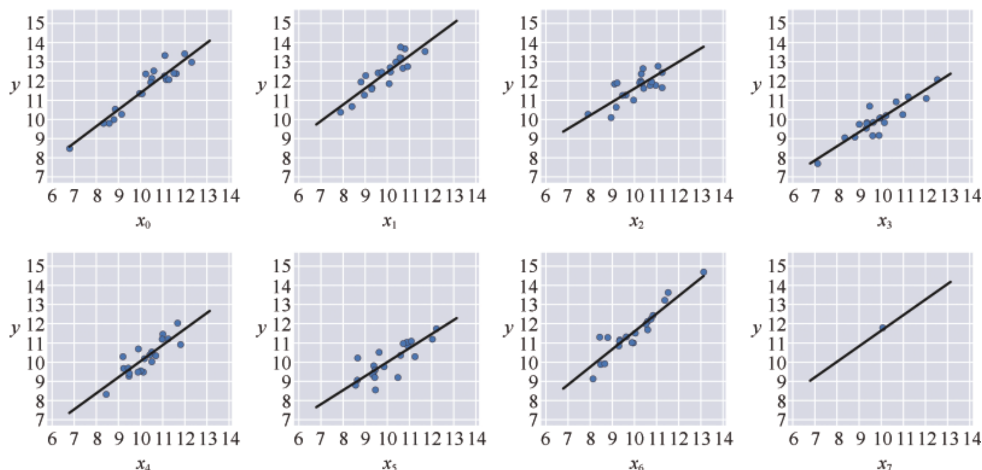
接下来是 `traceplot` 部分，包括只有一个点的组。

```
varnames=['alpha', 'alpha_mu', 'alpha_sd', 'beta', 'beta_mu', 'beta_sd',
'epsilon', 'nu']
pm.traceplot(trace_hm, varnames)
```



将拟合的直线画出来，包括只有一个点的那一组。显然，那条直线主要受到了其他组数据点的影响。

```
  j, k = 0, N
  x_range = np.linspace(x_m.min(), x_m.max(), 10)
for i in range(M):
     plt.subplot(2,4,i+1)
     plt.scatter(x_m[j:k], y_m[j:k])
     alfa_m = cadena_a['alpha'][:,i].mean()
     beta_m = cadena_a['beta'][:,i].mean()
     plt.plot(x_range, alfa_m + beta_m*x_range, c='k', label='y = 
{:.2f} + {:.2f} * x'.format(alfa_m, beta_m))
     plt.xlim(x_m.min()-1, x_m.max()+1)
     plt.ylim(y_m.min()-1, y_m.max()+1)
     j += N
     k += N
 plt.tight_layout()
```



4.3.1 相关性与因果性

现在假设已经知道了当地的太阳辐射量，我们想要预测冬天家里的燃气费。在这个问题中，太阳的辐射量是自变量 x，燃气费是因变量 y。当然，我们完全可以将问题反过来，根据燃气费推算太阳辐射量，一旦我们建立了一种线性关系（或者其他什么关系），我们就可以根据 x 得出 y，或者反过来这么做。我们称一个变量为自变量是因为它的值不是从模型中预测出来的，而是作为模型的输入，相应的因变量作为模型的输出。当我们说一个变量依赖于另一个变量的时候，这其中的依赖关系是由模型决定的。

我们建立的并不是变量之间的因果关系，即并不是说 x 导致了 y。永远要记住这句话：相关性并不意味着因果关系。就这个话题多说一点，我们可能根据家中的燃气费预测出太阳辐射量或者反过来根据太阳辐射量预测出家中的燃气费。但是我们显然并不能通过调节燃气阀门来控制太阳的辐射量。不过，太阳辐射量的高低是与燃气费的高低相关的。

因此，需要强调一点，我们构建的统计模型是一回事，变量之间的物理机制又是另外一回事。想要将相关性解释为因果关系，我们还需要给问题的描述增加一些可信的物理机制，仅仅相关性还不够。有一个网页，描述了一些有相关性但并没有因果关系的变量：http://www.tylervigen.com/spurious-correlations。

那么，相关性是否在确定因果关系时一点用都没有呢？不是。事实上，如果能够进行一些精心设计的实验，那么相关性是能够用于支撑因果关系的。举例来说，我们知道全球变暖与大气中二氧化碳的含量是高度相关的。仅仅根据这个观测，我们无法得出结论是温度升高导致的二氧化碳含量上升，还是二氧化碳含量的上升导致了温度升高。更进一步，可能存在某种我们没考虑到的第 3 个变量，导致二氧化碳含量和温度同时上升了。不过，我们可以设计一个实验，将玻璃箱子中充满不同比例的二氧化碳含量，其中一个是正常空气中的含量（约 0.04%），其余箱子中二氧化碳含量逐渐增加，然后让这些箱子接受一定时间的阳光照射（比如 3 个小时）。如果这么做之后能证实二氧化碳含量较高的箱子温度也更高，那么就能得出二氧化碳的含量导致温室效应的结论。同样的实验，我们可以反过来让相同二氧化碳含量的箱子接受不同温度的照射，然后可以看到二氧化碳含量并不会上升（至少空气中的二氧化碳含量不会上升）。事实上，更高的温度会导致二氧化碳含量的上升，因为海洋中蕴含着二氧化碳，随着温度上升，水中蕴含的二氧化碳含量会降低。简言之，全球正在变暖而我们没有采取足够措施解决该问题。

这个例子中还有一点需要说明下，尽管太阳辐射量与燃气费相关，根据太阳辐射量可能预测出燃气费，不过如果考虑到一些其他变量，这中间的关系就变得复杂了。我们一起来看一下，更高的太阳辐射量意味着更多的能量传递到家里，部分能量被反射掉了，还有部分转化成了热能，其中部分热量被房子吸收，还有部分散失到环境中了。热能消失多少取决于许多因素，比如室外的温度、

风力等。此外，我们还知道，燃气费也受到很多因素影响，比如国际上石油和燃气的价格，燃气公司的成本 / 利润（及其贪婪程度），国家对燃气公司的管控等。而我们在尝试用两个变量和一条直线对所有这一切建模。因此，充分考虑问题的上下文是有必要的，而且有利于得出更合理的解释，降低得出荒谬结论的风险，从而得到更好的预测，此外还有可能为我们提供线索改进模型。

4.4 多项式回归

接下来，我们将学习如何用线性回归拟合曲线。使用线性回归模型去拟合曲线的一种做法是构建如下多项式：

$$\mu = \beta_0 \boldsymbol{x}^0 + \beta_1 \boldsymbol{x}^1 + \beta_2 \boldsymbol{x}^2 + \beta_3 \boldsymbol{x}^3 + \cdots + \beta_n \boldsymbol{x}^n$$

如果留心的话，可以看到多项式中其实包含了一元线性回归模型，只需要将 n 大于 1 的系数 β_n 设为 0 即可，然后得到下式：

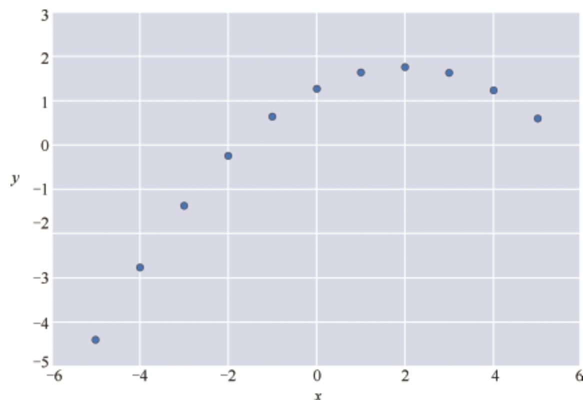
$$\mu = \beta_0 \boldsymbol{x}^0 + \beta_1 \boldsymbol{x}^1$$

多项式回归仍然是线性回归，这里线性的意思是指模型中的参数是线性组合的，而不是指变量是线性变化的。现在我们先从一个简单的多项式（抛物线）开始构建多项式回归模型：

$$\mu = \beta_0 \boldsymbol{x}^0 + \beta_1 \boldsymbol{x}^1 + \beta_2 \boldsymbol{x}^2$$

其中第 3 项控制的是曲率。我们选用安斯库姆四重奏的第 2 组作为数据集，通过 seaborn 将其载入并画出来。

```
ans = sns.load_dataset('anscombe')
x_2 = ans[ans.dataset == 'II']['x'].values
y_2 = ans[ans.dataset == 'II']['y'].values
x_2 = x_2 - x_2.mean()
y_2 = y_2 - y_2.mean()
plt.scatter(x_2, y_2)
plt.xlabel('$x$', fontsize=16)
plt.ylabel('$y$', fontsize=16, rotation=0)
```

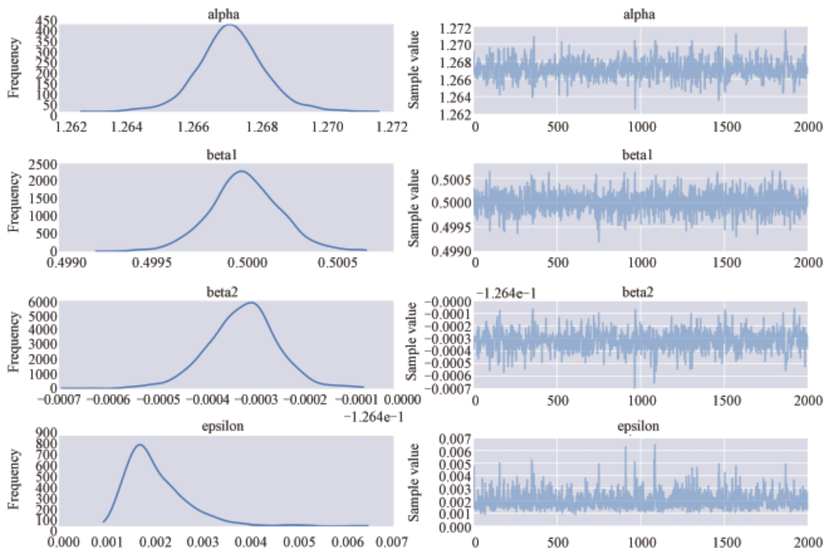



```
with pm.Model() as model_poly:
    alpha = pm.Normal('alpha', mu=0, sd=10)
    beta1 = pm.Normal('beta1', mu=0, sd=1)
    beta2 = pm.Normal('beta2', mu=0, sd=1)
    epsilon = pm.Uniform('epsilon', lower=0, upper=10)

    mu = alpha + beta1 * x_2 + beta2 * x_2**2

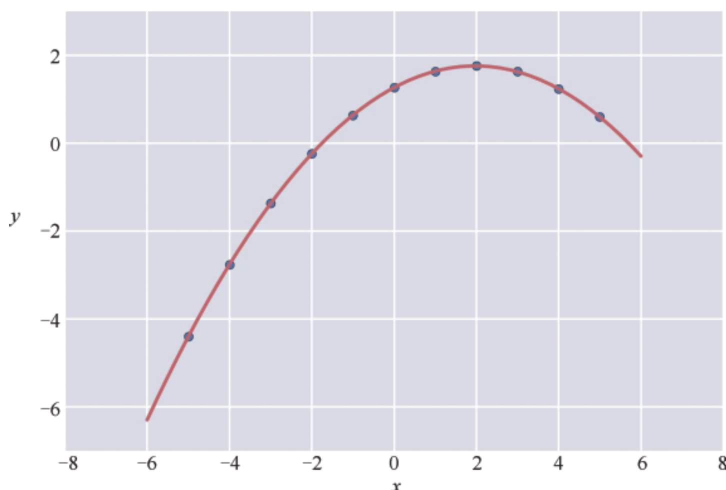
    y_pred = pm.Normal('y_pred', mu=mu, sd=epsilon, observed=y_2)

    start = pm.find_MAP()
    step = pm.NUTS(scaling=start)
    trace_poly = pm.sample(3000, step=step, start=start)
pm.traceplot(trace_poly)
```

这里再次省略了一些检查和总结，只将结果画了出来，可以看到一条非常好看的曲线完美拟合了数据，几乎没有误差。

```
x_p = np.linspace(-6, 6)
y_p = trace_poly['alpha'].mean() + trace_poly['beta1'].mean() * x_p + trace_poly['beta2'].mean() * x_p**2
plt.scatter(x_2, y_2)
plt.xlabel('$x$', fontsize=16)
plt.ylabel('$y$', fontsize=16, rotation=0)
plt.plot(x_p, y_p, c='r')
```



4.4.1　解释多项式回归的系数

多项式回归的问题之一在于参数的可解释性。如果我们想知道 y 相对于 x 的变化量，不能只看 β_1，因为 β_2 和更高项的系数对其也有影响。因此，系数 β 的值不再表示斜率。前面的例子中 β_1 是正数，因而曲线是以一个大于 0 的斜率开始的，不过由于 β_2 是负数，因而随后曲线的斜率开始下降。这看起来就好像有两股力量，一个使直线向上，另一个使直线向下，二者相互作用的结果取决于 x，当 $x_i < \sim 11$ 时，β_1 起决定作用，而当 $x_i > \sim 11$ 时，β_2 起决定作用。

如何解释参数不仅仅是个数学问题，因为我们可以通过仔细检查和理解模型来解决这个问题。不过许多情况下，参数并不能根据我们的领域知识转换成有意义的量，我们无法将其与细胞的新陈代谢速率或者恒星释放的能量或者房间里

的卧室数联系起来。它们只是些没有物理意义的参数。这样一个模型或许对于预测很有用，不过对于理解数据在底层是如何生成的并没有多大帮助。而且在实际中，超过 2 阶或者 3 阶的多项式模型并没有多大用途，我们更倾向于使用一些其他模型，这部分将在后面的章节中讨论。

4.4.2　多项式回归——终极模型?

我们知道，直线可以看作是当 β_2 为 0 时的抛物线的子模型，还可以看作是 β_2 和 β_3 都为 0 时的 3 次方模型的子模型。显然，抛物线模型也可以看作是当 β_3 为 0 时 3 次方模型的子模型……这意味着有一种算法可以使用线性回归模型去拟合任意复杂的模型，我们先构建一个无限高阶的多项式，然后将其中的大部分参数置零，直到我们得到对数据的完美拟合。为了验证这个想法，你可以从简单的例子开始，用刚刚构建的 2 次模型去拟合安斯库姆四重奏的第 3 个数据集。

完成练习之后，你会发现用 2 次模型去拟合直线是可能的。这个例子看起来似乎验证了可以使用无限高阶多项式去拟合数据这一思想，但是通常用多项式去拟合数据并不是最好的办法。为什么呢？因为它并不关心具体使用的数据是怎么来的，从原理上讲，我们始终可以找到一个多项式去完美拟合数据。如果一个模型完美拟合了当前的数据，那么通常对于没有观测到的数据会表现得很糟糕，原因是现实中的任意数据集都同时包含一些噪声和一些感兴趣的模式。一个过于复杂的模型会拟合噪声，从而使得预测的结果变差，这称作过拟合，一个在统计学和机器学习中常见的现象。越复杂的模型越容易导致过拟合，因而分析数据时，需要确保模型没有产生过拟合，我们将在第 6 章模型比较中详细讨论。

除了过拟合问题，我们通常更倾向于更容易理解的模型。从物理意义上讲，线性模型的参数要比 3 次模型的参数更容易解释，即便 3 次模型对数据拟合得更好。

4.5　多元线性回归

前面的所有例子中，我们讨论的都是一个因变量和一个自变量的情况，不过

在许多例子中，我们的模型可能包含多个自变量。例如：

- 红酒的口感（因变量）与酒的酸度、比重、酒精含量、甜度以及硫酸盐含量（自变量）的关系；
- 学生的平均成绩（因变量）与家庭收入、家到学校的距离、母亲的受教育程度（自变量）的关系。

这种情况下，因变量可以这样建模：

$$\mu = \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_m x_m$$

注意这个式子与前面看到过的多项式回归的式子不一样，现在有了多个变量而不再是一个变量的多次方。 用线性代数的表示方法可以写成更简洁的形式：

$$\mu = \alpha + \boldsymbol{\beta X}$$

其中，$\boldsymbol{\beta}$ 是一个长度为 m 的系数向量，也就是说，自变量的个数为 m。变量 $\boldsymbol{X}$ 是一个维度为 $n \times m$ 的矩阵，其中，n 表示观测的样本数，m 表示自变量个数。有关线性代数，你可以阅读维基百科上关于向量点乘和矩阵乘法的相关知识，可以用以下这种更简洁的形式描述模型：

$$\boldsymbol{\beta X} = \sum \beta_i x_i = \beta_1 x_1 + \cdots + \beta_m x_m$$

在一元线性回归模型中，我们希望找到一条直线来解释数据，而在多元线性回归模型中，我们希望找到的是一个维度为 m 的超平面。因此，多元线性回归模型本质上与一元线性回归模型是一样的，唯一的区别是：现在 $\boldsymbol{\beta}$ 是一个向量而 $\boldsymbol{X}$ 是一个矩阵。 现在我们定义如下数据：

```
np.random.seed(314)
N = 100
alpha_real = 2.5
beta_real = [0.9, 1.5]
eps_real = np.random.normal(0, 0.5, size=N)

X = np.array([np.random.normal(i, j, N) for i,j in zip([10, 2], [1, 1.5])])
X_mean = X.mean(axis=1, keepdims=True)
X_centered = X - X_mean
y = alpha_real + np.dot(beta_real, X) + eps_real
```

然后定义一个函数去画 3 个散点图，前两个表示的是自变量与因变量的关

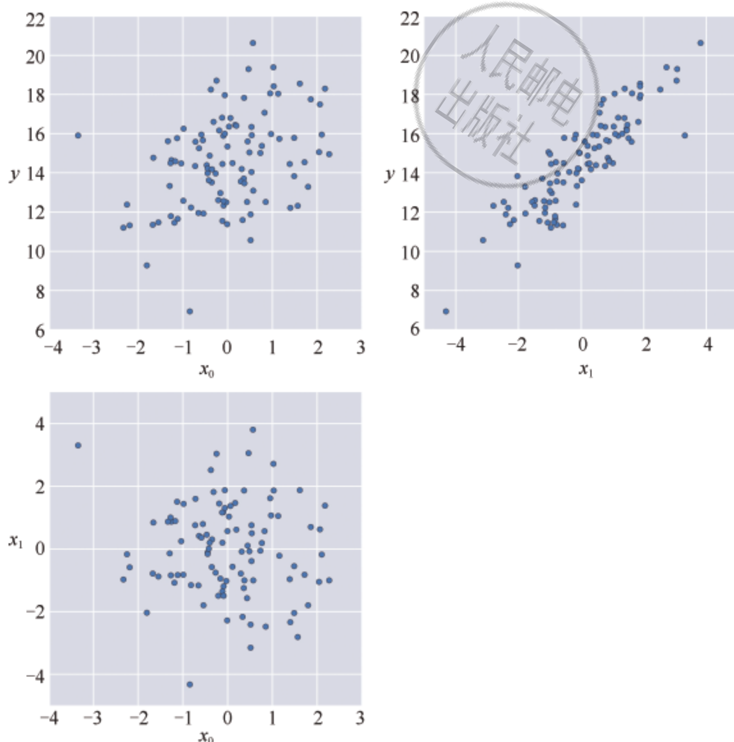
系，最后一个表示的是两个自变量之间的关系。这只是个普通的绘图函数，本章后面将会反复用到。

```
def scatter_plot(x, y):
    plt.figure(figsize=(10, 10))
    for idx, x_i in enumerate(x):
        plt.subplot(2, 2, idx+1)
        plt.scatter(x_i, y)
        plt.xlabel('$x_{}$'.format(idx), fontsize=16)
        plt.ylabel('$y$', rotation=0, fontsize=16)

    plt.subplot(2, 2, idx+2)
    plt.scatter(x[0], x[1])
    plt.xlabel('$x_{}$'.format(idx-1), fontsize=16)
    plt.ylabel('$x_{}$'.format(idx), rotation=0, fontsize=16)
```

用前面刚刚定义的 scatter_plot 可以将我们的合成数据可视化地表示出来。

```
scatter_plot(X_centered, y)
```



现在用 PyMC3 针对多变量线性回归问题定义出一个合适的模型，代码部分

与单变量线性回归的代码基本一致，唯一的区别是：

- 参数 beta 是高斯分布，大小为 2，每个参数都有一个斜率；
- 这里使用 `pm.math.dot()` 来定义变量 mu，也就是前面提到的线性代数中的点乘（或者矩阵相乘）。

如果你对 NumPy 比较熟悉，那么你应该知道 NumPy 包含一个点乘函数，而且 Python 3.5（以及 NumPy 1.10）之后增加了一个新的操作符 `@`。不过这里我们使用的是 PyMC3 中的点乘函数（其实是 Theano 中矩阵相乘的一个别名），因为变量 beta 在这里是一个 Theano 中的张量而不是 NumPy 数组。

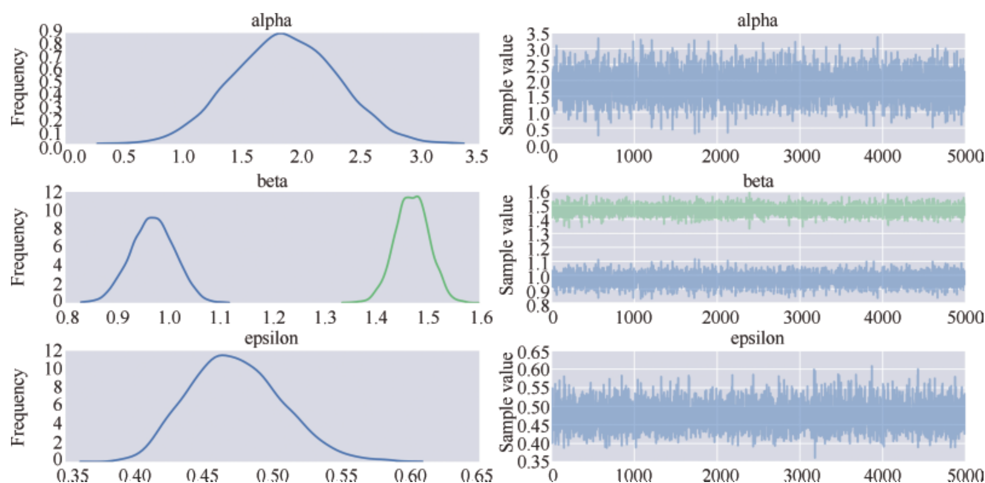
```
with pm.Model() as model_mlr:
    alpha_tmp = pm.Normal('alpha_tmp', mu=0, sd=10)
    beta = pm.Normal('beta', mu=0, sd=1, shape=2)
    epsilon = pm.HalfCauchy('epsilon', 5)

    mu = alpha_tmp + pm.math.dot(beta, X_centered)

    alpha = pm.Deterministic('alpha', alpha_tmp - pm.math.dot(beta, X_mean))

    y_pred = pm.Normal('y_pred', mu=mu, sd=epsilon, observed=y)

    start = pm.find_MAP()
    step = pm.NUTS(scaling=start)
    trace_mlr = pm.sample(5000, step=step, start=start)
varnames = ['alpha', 'beta', 'epsilon']
pm.traceplot(trace_mlr, varnames)
```



现在看一下推断出来的参数的总结，这样分析结果会更容易一些。我们的模型表现如何呢？

```
pm.df_summary(trace_mlr, varnames)
```

	mean	sd	mc_error	hpd_2.5	hpd_97.5
alpha_0	2.07	0.50	5.96e-03	1.10	3.09
beta_0	0.95	0.05	6.04e-04	0.85	1.05
beta_1	1.48	0.03	4.30e-04	1.43	1.54
epsilon	0.47	0.03	4.63e-04	0.41	0.54

可以看到，模型能够重现正确的值（对比生成数据用的值）。接下来，我们将重点关注在分析多变量线性回归模型中需要注意的点，特别是对斜率的解释。这里需要特别提醒的是：每个参数只有在整体考虑了其他参数的情况下才有意义。

4.5.1　混淆变量和多余变量

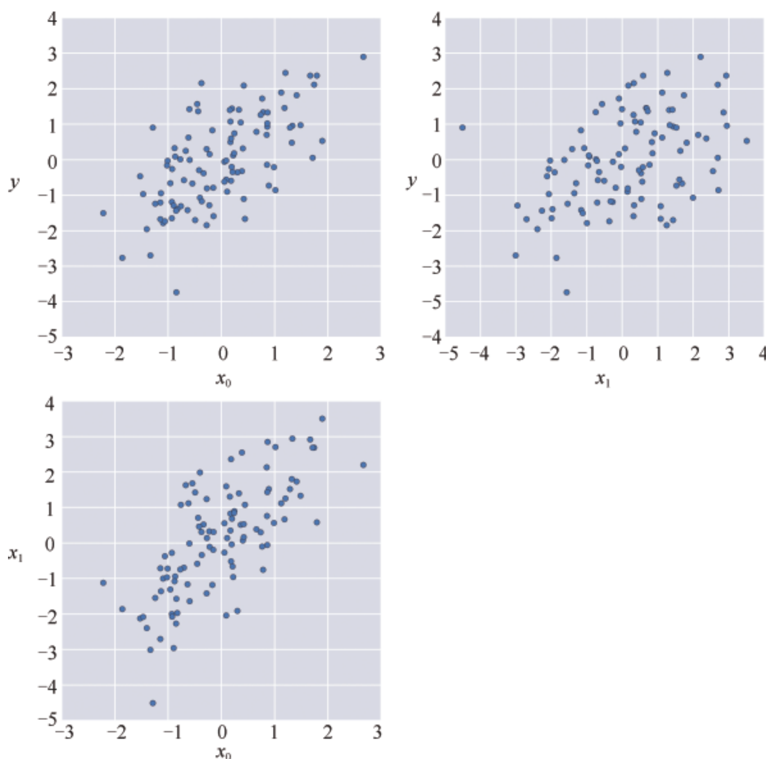
设想这样一种情况：有一个变量 z 与预测变量 x 相关，同时还与另一个被预测的变量 y 相关。假设 z 对 x 和 y 都有影响，例如，z 可以是工业革命（一个相当复杂的变量），x 是海盗的数量，y 是二氧化碳浓度。如果在分析中将 z 去掉，我们会得出结论：x 与 y 之间有着完美的线性相关性，我们甚至可以通过 x 预测 y。不过，如果我们关注的重点是如何缓解全球变暖问题，那么可能完全没搞清到底发生了什么以及其内在机制是什么。记住，前面已经讨论了相关性并不意味着因果关系，原因是我们可能在分析过程中忽略了变量 z。这种情况下，z 称作混淆变量，或者是混淆因素。问题是在很多情况下，z 很容易被忽略。也许是因为我们压根没有测量 z，或者是因为没有包含在传给我们的数据集中，又或者是因为我们压根没想到它可能与我们的问题有联系。没有考虑到混淆变量可能会导致我们的分析得出奇怪的相关性，在解释数据和做预测（有时候我们不关心内在的机制）的时候，这可能是个问题。理解底层的机制有利于将学到的东西迁移到新的场景中，相反，盲目的预测很难迁移。例如，帆布鞋产量可以作为一个国家经济实力的易测指标，不过对于生产链不同或者文化背景不同的国家而言，这可能是个糟糕的指标。

我们将使用合成数据来探索混淆变量的问题。下面的代码中模拟了一个混淆变量 x_1，注意这个变量是如何影响 x_2 和 $\boldsymbol{y}$ 的。

```
N = 100
x_1 = np.random.normal(size=N)
x_2 = x_1 + np.random.normal(size=N, scale=1)
y = x_1 + np.random.normal(size=N)
X = np.vstack((x_1, x_2))
```

根据生成数据的方式，可以看出这些变量已经是中心化了。因此，不需要再进一步对数据进行中心化处理来加速推断过程了。事实上这个例子中的数据已经是标准化的了。

```
scatter_plot(X, y)
```



然后用 PyMC3 创建模型并从中采样，现在，你对这个模型应该已经相当熟悉了。

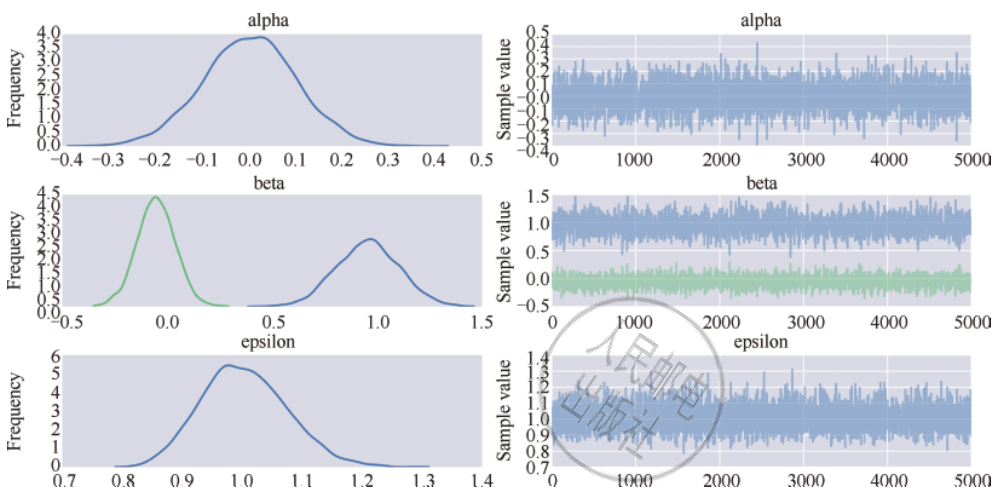
```
with pm.Model() as model_red:
    alpha = pm.Normal('alpha', mu=0, sd=10)
```

```
        beta = pm.Normal('beta', mu=0, sd=10, shape=2)
        epsilon = pm.HalfCauchy('epsilon', 5)

        mu = alpha + pm.math.dot(beta, X)

        y_pred = pm.Normal('y_pred', mu=mu, sd=epsilon, observed=y)

        start = pm.find_MAP()
        step = pm.NUTS(scaling=start)
        trace_red = pm.sample(5000, step=step, start=start)
```



现在用 pandas 中的 dataframe 将结果的总结打印出来，重点关注 beta 参数的均值。

```
pm.df_summary(trace_red)
```

	mean	sd	mc_error	hpd_2.5	hpd_97.5
alpha	0.01	0.10	1.59e-03	−0.20	0.19
beta_0	0.96	0.16	3.13e-03	0.67	1.29
beta_1	−0.05	0.10	2.06e-03	−0.24	0.14
epsilon	1.00e+00	0.07	1.15e-03	0.87	1.15

可以看到 β_1 接近 0，这意味着 x_2 对 $\boldsymbol{y}$ 来说几乎没有作用。因为通过检查合成的数据已经知道了起重要作用的是变量 x_1。现在需要先做一些测试，重跑两次模型，其中一次只用 x_1，另一次只用 x_2。如果你查看下代码的话，可以看到有几行是被注释了的，你或许会用得上。请问，这 3 种情况下 beta 系数的均值有多

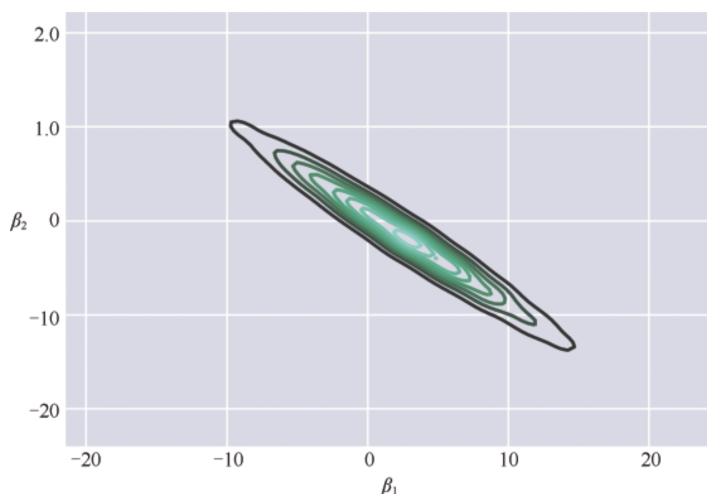
大区别？如果你做了实验，就会注意到，对于 β_2 的值，多元线性回归模型中得到的要比一元线性回归模型得到的更低。换句话说，当 x_1 加入到模型中之后，x_2 对模型的解释性降低了（甚至降为 0）。

4.5.2 多重共线性或相关性太高

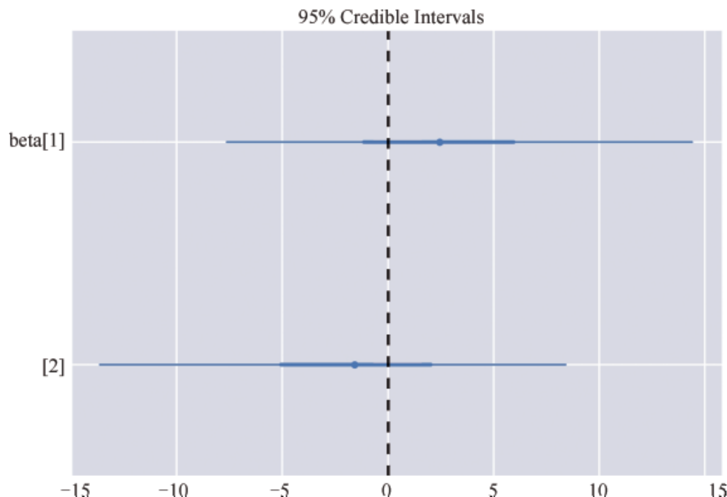
前面的例子中，我们看到了多元线性回归模型是如何处理冗余变量的，同时还了解了混淆变量的重要性。接下来我们沿着前面的例子继续深入学习当两个变量高度相关时会发生什么。为了研究这个问题及其对推断的影响，我们使用和前面一样的合成数据和模型，不过通过减小根据 x_1 生成 x_2 时的随机噪声，增加了 x_1 和 x_2 之间的相关性：

```
x_2 = x_1 + np.random.normal(size=N, scale=0.01)
```

上面这行代码相当于对 x_1 增加了一个很小的扰动，因而得到的两个变量可以看作是一样的，然后你可以修改下数据的尺度并引入少量的极限值，不过，当下我们尽量保持简单。生成新的数据后，可以用散点图查看下数据是什么样的；你应该可以看到 x_1 与 x_2 之间的关系是一条斜率接近于 1 的直线。运行模型并对结果进行检查。在本书附带的代码中，你可以看到有几行代码可以根据 beta 系数画 2D 的核密度估计图。你应该可以看到与下面类似的图。



beta 参数的 HPD 区间相当广，与先验几乎一样。



可以看到，beta 的后验是一条很窄的斜对角线。当其中一个 beta 参数增加时，另一个一定下降。两个参数非常相关，这是模型和数据共同作用的结果。根据我们的模型，均值 μ 有如下形式：

$$\mu=\alpha+\beta_1 x_1+\beta_2 x_2$$

假设 x_1 和 x_2 不只是近似相同，而是完全一样的，那么可以将模型改写成如下形式：

$$\mu=\alpha+(\beta_1+\beta_2)\boldsymbol{x}$$

可以看到，对 μ 有影响的是 β_1 与 β_2 的和而不是二者单独的值，因而模型是不确定的（或者说，数据并不能决定 β_1 和 β_2 的值）。在我们的例子中，beta 并不能在区间 [- ∞ , ∞] 内自由移动，原因有两个：其一，两个变量几乎是相同的，不过并非完全一样；其二，更重要的是 beta 系数的可能取值受到先验的限制。

这个例子中有几点需要注意。首先，后验只是根据模型和数据得出的逻辑上的结果，因而得出一个分布很广的 beta 分布并没有错，事实就是这样子；第 2 点是，我们可以依据该模型做预测，可以尝试做一些后验预测检查，该模型预测得到的值与数据分布是一致的，也就是说模型对数据拟合得很好；

第 3 点是，对于理解问题而言这可能不是一个很好的模型，更好的做法是从模型中去掉一个参数，这样模型的预测能力与以前一样，但是更容易解释。

在任何真实的数据集中，相关性在某种程度上是普遍存在的。那么两个或多个变量之间相关性多高时会导致问题呢？事实上并没有确切的数值。我们可以在运行贝叶斯模型之前，构建一个相关性矩阵，对其中相关性较高（比如说高于 0.9）的变量进行检查。不过，这种做法的问题是：根据相关性矩阵观察到的成对变量之间的相关性并不太重要，重要的是在某个具体模型中变量的相关性。前面已经看到了，不同变量在单独情况下的表现与在模型中放一起的表现是不同的。在多元回归模型中，两个或多个变量之间的相关性可能会受到其他变量的影响，从而使得他们之间的相关性降低或者升高。通常，建议在迭代式构建模型的同时加入一些诊断（比如检查自相关性和后验），这有利于发现问题和理解模型与数据。

如果发现了高度相关的变量应该怎么做呢？

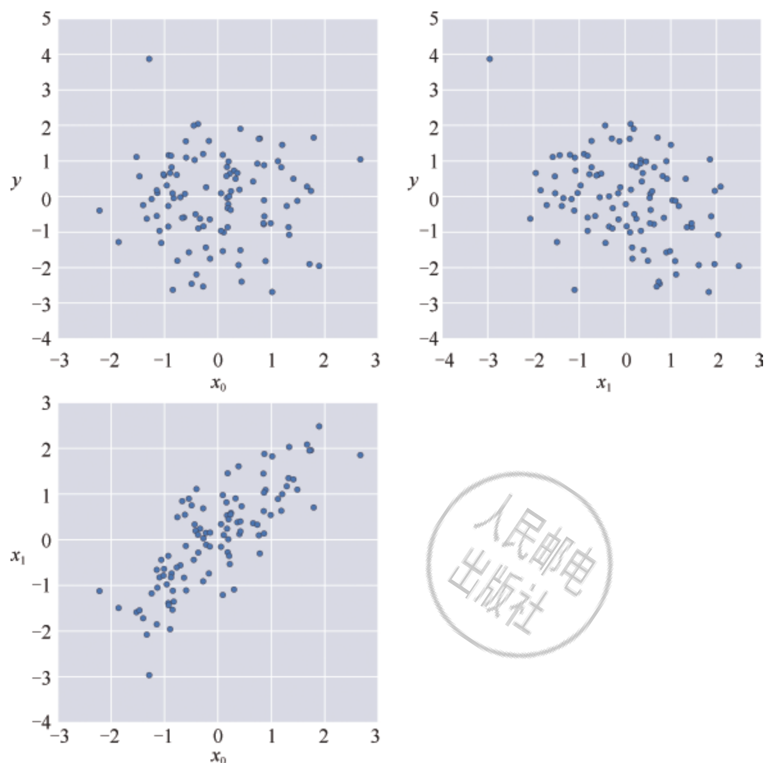
- 如果相关性非常高，我们可以从分析中将其中一个变量去掉。如果两个变量的信息都差不多，具体去掉哪个并不重要，可以视方便程度（比如去掉最不常见的或者最难解释或测量的变量）。
- 另外一种可行的做法是构建一个新的变量对冗余变量求均值。更高级的做法是使用一些降维算法，如主成分分析法（PCA）。不过 PCA 的一个问题是得到的结果变量是原始变量的线性组合，通常会对结果的可解释性造成模糊。
- 还有一种办法是给变量可能的取值设置一个较强的先验。在第 6 章中我们会简要讨论如何选择这类先验（通常称作正则先验）。

4.5.3 隐藏的有效变量

有一种情况与前面见过的类似，其中某个变量与预测变量正相关而另外一个与预测变量负相关。这里先构建一些玩具数据来说明。

```
N = 100
r = 0.8
x_0 = np.random.normal(size=N)
```

```
x_1 = np.random.normal(loc=x_0 * r, scale=(1 - r ** 2) ** 0.5)
y = np.random.normal(loc=x_0 - x_1)
X = np.vstack((x_0, x_1))
scatter_plot(X, y)
```

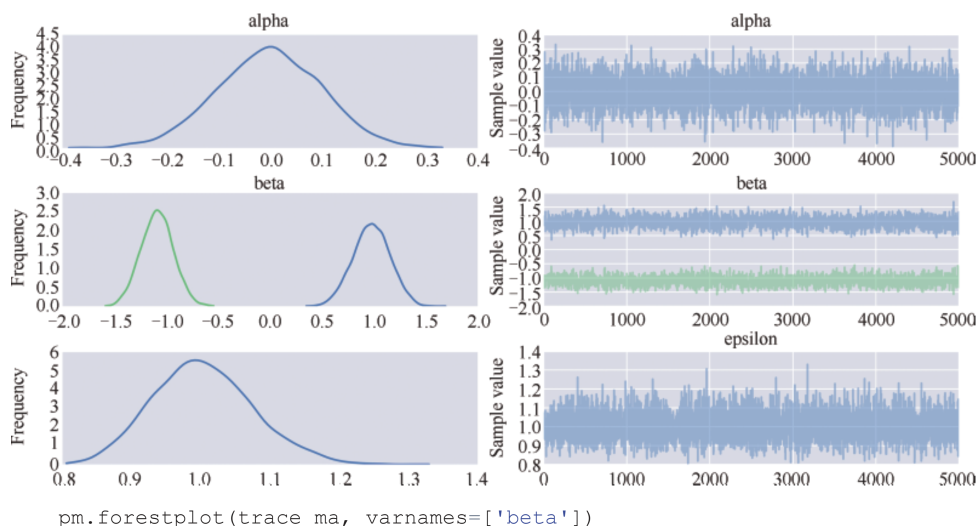



```
with pm.Model() as model_ma:
    alpha = pm.Normal('alpha', mu=0, sd=10)
    beta = pm.Normal('beta', mu=0, sd=10, shape=2)
    epsilon = pm.HalfCauchy('epsilon', 5)

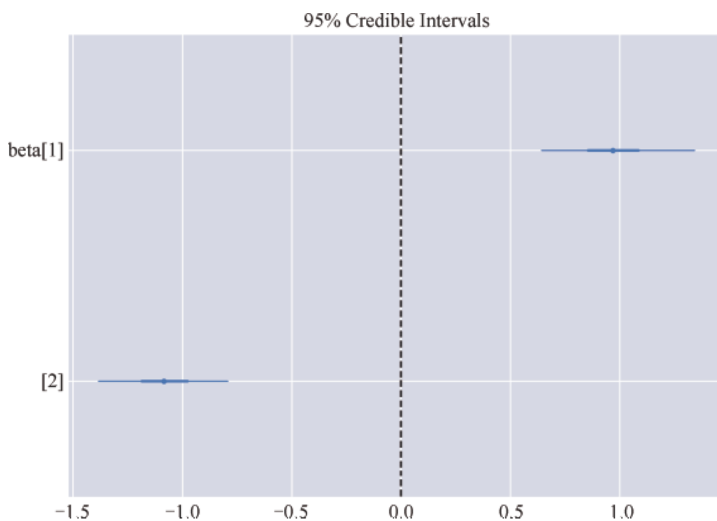
    mu = alpha + pm.math.dot(beta, X)

    y_pred = pm.Normal('y_pred', mu=mu, sd=epsilon, observed=y)

    start = pm.find_MAP()
    step = pm.NUTS(scaling=start)
    trace_ma = pm.sample(5000, step=step, start=start)
pm.traceplot(trace_ma)
```




```
pm.forestplot(trace_ma, varnames=['beta'])
```



从后验可以看出，beta 的值接近 1 和 −1。也就是说，x_1 与 $\boldsymbol{y}$ 正相关而 x_2 与 $\boldsymbol{y}$ 负相关。现在重新分析，不过（也许你已经猜到了）这一次我们对每个单独的变量进行分析。

对于单个的变量，可以看到 $\boldsymbol{\beta}$ 接近 0，也就是说每个单独变量 $\boldsymbol{x}$ 都不足以预测 $\boldsymbol{y}$。相反，如果我们将 $\boldsymbol{x}$ 组合在一起后就可以预测 $\boldsymbol{y}$。当 x_1 增加时 x_2 也增加，而当 x_2 增加时 $\boldsymbol{y}$ 也降低，因此如果单独看变量 x_1 而忽略 x_2 的话，我们会看到当 x_1 增加的时候，$\boldsymbol{y}$ 几乎不增加，而当 x_2 增加时，$\boldsymbol{y}$ 几乎不降低。因变量之间具有

相关性，每个因变量都有反作用，因而忽略其中任何一个都会造成对变量影响力的低估。

4.5.4　增加相互作用

目前为止，所有多元回归模型的定义中，在其他预测变量固定的条件下，x_1 的变化都会（隐式地）带来 $\boldsymbol{y}$ 的稳定变化。不过这显然并非一定的，有可能改变 x_2 之后，原来 $\boldsymbol{y}$ 与 x_1 之间的关系发生了改变。一个经典的例子是药物之间的相互作用，例如，在没有使用药物 B（或者药物 B 的剂量较低）时，增加药物 A 的剂量有正向影响，而当增加药物 B 的剂量时，药物 A 反而有负向（甚至致命的）影响。

目前见过的所有例子中，因变量对于预测变量的作用都是叠加的。我们做的只是增加变量（每个变量乘以一个系数）。如果我们希望捕捉到变量的效果，就像前面的药物例子一样，我们需要给模型增加一项非叠加的量，比如，变量之间的乘积：

$$\mu=\alpha+\beta_1x_1+\beta_2x_2+\beta_3x_1x_2$$

注意这里系数 β_3 乘的是 x_1 和 x_2 的乘积，这个非叠加项只是一个用来说明统计学中的变量之间相互作用的例子，因为它衡量了变量（在前面的例子中是药物）之间的相关性。对相关性建模的表达式有很多种，相乘只是其中一个比较常用的。

在多元线性回归模型中，如果没有变量之间的乘积，我们得到的是一个超平面，也就是说，一个平坦的超曲面，加入乘积之后，该超曲面会变得弯曲。

4.6　glm 模块

在统计学和机器学习中，线性模型有着广泛的应用，PyMC3 有一个叫做 glm 的模块，也就是通用线性模块（generalized linear model）。一个线性模型可以这么表示：

```
with Model() as model:
    glm.glm('y ~ x', data)
```

```
    trace = sample(2000)
```

代码中的第 2 行会默认给截距和斜率添加均匀先验，同时给因变量添加一个高斯似然。如果你只是想跑一个默认的线性回归问题，这样就可以了，需要注意这个模型的 MAP 在本质上跟使用普通的（频率学派中的）最小二乘法得到的结果是一样的。如果需要，你可以使用 glm 模块并修改其中的先验和似然。如果你对 R 语言的语法不熟，需要说明一下，这里 `y ~ x` 的意思是用 glm 模块中的线性函数定义了输出变量 y。在 glm 模块中还包含一个函数用来画后验预测的图。

4.7 总结

线性回归是统计学和机器学习中最常用的模型之一，同时也是构建其他更复杂模型的基石，其应用相当广泛，不同领域的人对同一个东西有不同的称呼，因此本章一开始先介绍了统计学和机器学习中的一些常见词汇。然后深入研究了线性模型的核心，用一个表达式衔接了输入变量与输出变量。本章用到的是高斯分布和 t 分布作为因变量的似然，后面我们会将该模型扩展到其他分布。我们还处理了一些计算方面的问题，以及如何通过中心化和（或）标准化数据解决问题，还有幸见识到了 NUTS 相比 Metropolis 的优势。

在一元线性模型中，我们应用了多层模型，同时探讨了如何使用多项式回归拟合曲线以及这类模型的一些问题。此外，还讨论了如何用多个参数构建线性回归模型，解释线性模型中的一些注意事项，下一章我们将学习如何扩展线性回归模型并用于分类问题。

4.8 深入阅读

- 《Statistical Rethinking》中的第 4 章和第 5 章。
- 《Doing Bayesian Data Analysis, Second Edition》中的第 17 章和第 18 章。
- 《An Introduction to Statistical Learning》中的第 4 章。
- 《Bayesian Data Analysis, Third Edition》中的第 14 章～第 17 章。
- 《Machine Learning: A probabilistic Probabilistic Perspective》中的第 7 章。
- 《Data Analysis Using Regression and Multilevel/Hierarchical Models》一书。

4.9　练习

（1）选一个你觉得有意思的数据集并用一元线性回归去拟合。用不同的方法重跑一遍，重新画图并计算出皮尔逊相关系数。如果没有合适的数据，可以上网找一下，网址为：http://data.worldbank.org/ 或者 http://www.stat.ufl.edu/~winner/datasets.html。

（2）阅读并运行 PyMC3 官方文档中的例子 https://pymc-devs.github.io/pymc3/notebooks/LKJ.html。

（3）对于没有进行池化操作的模型，尝试修改 beta 先验中 `sd` 的值（比如 1 和 100），观察每组数据中预估斜率值的变化，哪组数据更容易受到该变化的影响？

（4）查看本书附带的代码 `model_t2`（以及附带的数据），尝试更换 `nu` 的先验，比如非漂移的指数先验和伽马先验（代码中已经注释掉了）。画出先验并确保你理解了（一个简单的做法是将似然注释掉并运行 `traceplot` 函数）。

（5）降低 ADVI 的迭代次数（目前是 100000），例如降到 10000，对 NUTS 每秒迭代次数有什么影响？观察 `traceplot` 的返回结果，看看对采样值有什么影响。将 ADVI 换成前面其他模型用过的 `find_MAP()`，能否观察到这样做的优势？

（6）运行 `model_mlr` 的例子，不过这次不对数据进行中心化处理。比较两种情况下 alpha 参数的不确定性。你能对结果做出些解释吗？提示：回忆一下参数 alpha 的定义（截距）。

（7）阅读并运行以下 PyMC3 文档中的记事本：

- https://pymc-devs.github.io/pymc3/notebooks/GLM-linear.html
- https://pymc-devs.github.io/pymc3/notebooks/GLM-robust.html
- https://pymc-devs.github.io/pymc3/notebooks/GLM-hierarchical.html

（8）运行多元线性回归模型部分的练习。

第 5 章
利用逻辑回归对结果进行分类

上一章中，我们学习了线性回归模型的核心内容，在这类模型中，我们假设被预测的变量是定量的（连续变量）。这一章我们将学习如何处理定性的变量（或者称为类别变量），比如颜色、性别、生物种类、政党等。注意有些变量既可以看作是定性的又可以看作是定量的，比如红色和绿色，如果单看名字，它们是定性的，如果从它们的波长来看，又是定量的（分别为 650nm 和 510nm）。处理类型变量时一个常见的问题是对于某个观测值赋予类别标签，这类问题称为分类问题。分类问题属于有监督问题，因为我们已经有一些分类好的样本，只需要对新的样本预测其正确的类别，同时学习模型的参数用于描述特征到类别之间的映射关系。

本章将会讨论以下内容：

- 逻辑回归和逆连结函数；
- 一元逻辑回归；
- 多元逻辑回归；
- softmax 函数和多项逻辑回归。

5.1　逻辑回归

我母亲有一道拿手好菜叫 sopa seca，其做法源于意大利面，直译过来叫做**干汤面**，这个名字听起来似乎有点矛盾，不过一旦了解它的做法之后，你会觉得这个叫法其实挺合理的。同样，逻辑回归也是如此，它主要用来解决分类问题。逻辑回归模型是线性回归模型的一种扩展，这也是其名称来源。在理解如何将一个回归模型应用于分类问题之前，先将线性模型的核心部分改写成如下形式：

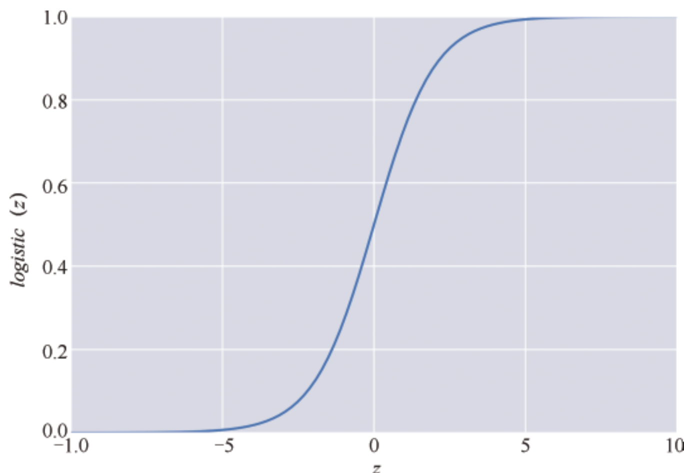
$$\mu = f(\alpha + \beta \boldsymbol{X})$$

其中，f 称作逆连结函数。为什么这里把 f 称作逆连结函数而不是连结函数？原因是传统上人们认为此类函数是用来连结输出变量和线性模型的，不过在构建贝叶斯模型的时候你将看到，反过来思考可能更容易一些。因此，为了避免疑惑，我们这里统称逆连结函数。前一章中的所有线性模型其实都包含一个逆连结函数，不过我们书写的时候将其省略了，因为它其实是一个恒等函数（函数的返回值和输入值相同）。恒等函数在这里也许没什么用，不过它可以让我们用一种更一般的形式思考不同的模型。从原理上讲，许多其他函数都可以充当逆连结函数，不过这一章只讨论逻辑函数，其数学形式如下：

$$logistic(z) = \frac{1}{1 + \exp(-z)}$$

从分类的角度来看，逻辑函数最重要的特点之一是不论参数 z 的值为多少，其输出值总是介于 0 到 1 之间。因此，该函数将整个实轴压缩到了区间 [0,1] 内。逻辑函数也称作 **S 型函数**（sigmoid function），因为它的形状看起来像 S，可以运行以下几行代码来看一下。


```
z = np.linspace(-10, 10, 100)
logistic = 1 / (1 + np.exp(-z))
plt.plot(z, logistic)
plt.xlabel('$z$', fontsize=18)
plt.ylabel('$logistic(z)$', fontsize=18)
```



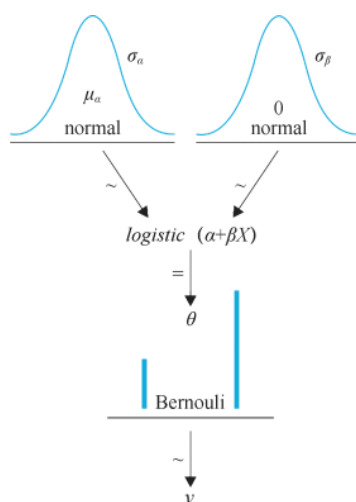
5.1.1 逻辑回归模型

现在我们知道逻辑函数长什么样了，接下来将继续学习如何用来对结果进行分类。先从简单的问题开始，假设类别只有两类，比如正常邮件 / 垃圾邮件、安全 / 不安全、阴天 / 晴天、健康 / 生病等。首先对类别进行编码，假设变量 y 只能有两个值，0 或 1，也就是说 $y \in \{0,1\}$。这样描述之后，问题就有点像抛硬币问题了，在那个例子中我们用到了伯努利分布作为似然。这里的区别是：现在 θ 不再是从一个 beta 分布中生成的，而是由一个线性模型定义的。一个线性模型可以返回实轴上的任意值，但是伯努利分布的值限定在 [0,1] 区间内。因此我们使用了一个逆连结函数将线性模型的返回值映射到一个适合伯努利分布的区间内，从而将一个线性回归模型转换成了分类模型：

$$\theta = logistic(\alpha + \beta \boldsymbol{X})$$

$$y \sim Bern(\theta)$$

下面的 Kruschke 图显示了逻辑回归模型（应该）包含的先验。注意与单参数线性回归模型的区别是：这里用到了伯努利分布而不是高斯分布（或者 t 分布），并使用了逻辑函数生成区间 [0,1] 范围内的参数 θ，从而适于输入到伯努利分布。



5.1.2 鸢尾花数据集

这里我们将逻辑回归应用到鸢尾花数据集上。在构建模型之前，我们先了解

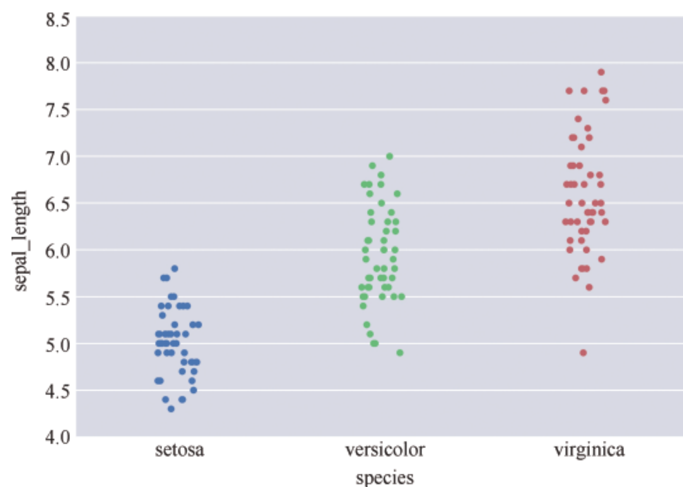
下该数据集。鸢尾花数据集是一个很经典的数据集，包含有 Setosa、Versicolour 和 Virginica 3 个种类，这 3 个类别标签就是我们想要预测的量，即因变量。其中每个种类包含有 50 个数据，每个数据包含 4 种变量（或者称为特征，机器学习中通常这么称呼），这 4 种变量就是我们要分析的自变量，分别是：花瓣长度、花瓣宽度、花萼长度和花萼宽度。花萼有点类似小叶，在花还是芽时包围着花，有保护作用。seaborn 中包含鸢尾花数据集，我们可以用如下代码将其导入成 Pandas 里的 Dataframe 格式：

```
iris = sns.load_dataset("iris")
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

可以用 seaborn 中的 stripplot 函数将每个种类的花萼长度画出来：

```
sns.stripplot(x="species", y="sepal_length", data=iris, jitter=True)
```

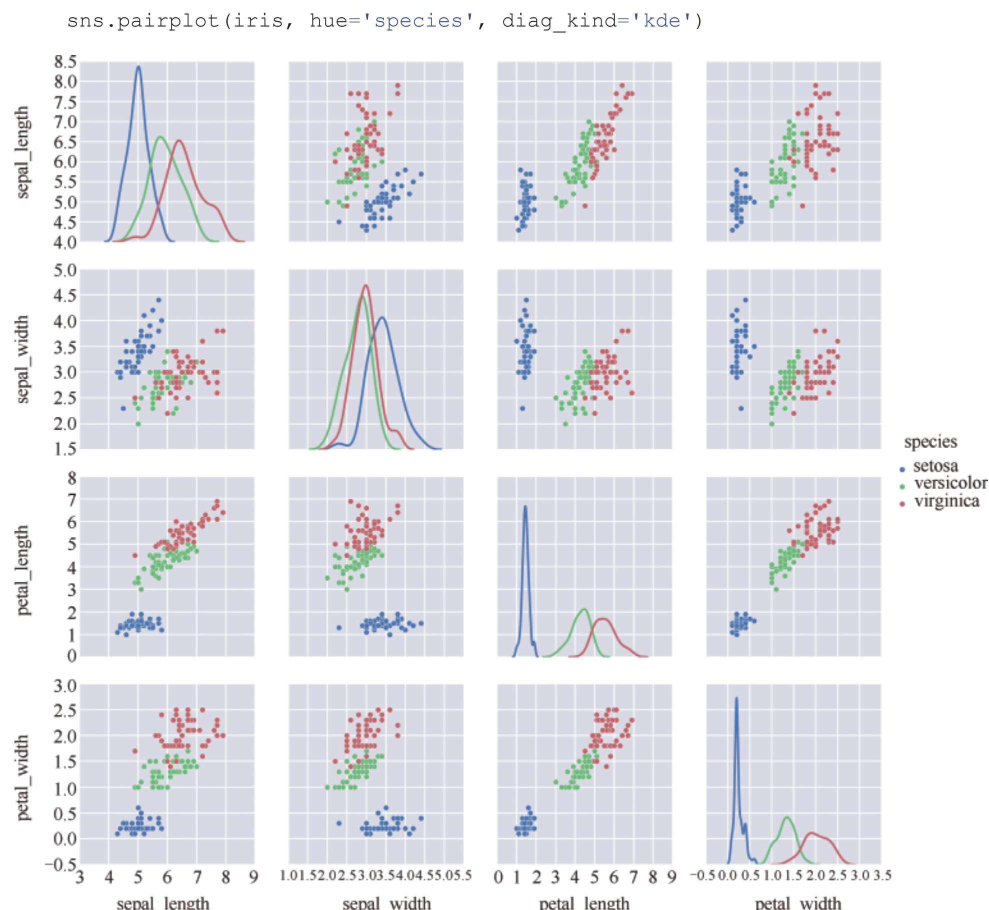


上图中 y 轴是连续的，而 x 轴是离散的类别变量；图中的点在 x 轴上是分散开来的，这并没有什么实际意义，只是一个画图的小技巧而已，通过将

jitter 参数设为 True，能够避免所有点都重叠在一条直线上，你可以试着将 jitter 参数设为 False。这里唯一重要的是 x 轴的含义，即分别代表 setosa、versicolour 和 virginica 3 个类别。你还可以用 seaborn 中的其他作图函数来画这些点（比如 violin plot），只需要一行代码就能完成。

另外一种观察数据的方法是用 pairplot 画出散点图矩阵，用该函数可以得到一个 4×4 的网格（因为我们有 4 种特征）。网格是对称的，上三角和下三角表示的是同样的信息。由于对角线上的散点图其实是变量本身，因此这里用一个特征的 KDE 图代替了散点图。可以看到，每个子图中，分别用 3 种颜色表示 3 种不同的类别标签，这一点与前面图中的表示一致。

```
sns.pairplot(iris, hue='species', diag_kind='kde')
```



深入学习之前，花点时间分析下前面这幅图，进一步熟悉这个数据集并了解变量与标签之间的关系。

5.1.3　将逻辑回归模型应用到鸢尾花数据集

我们先从一个简单的问题开始：用花萼长度这一特征（自变量）来区分 setosa 和 versicolor 这两个种类。和前面一样，这里用 0 和 1 对类别变量进行编码，利用 Pandas 可以这么做：

```
df = iris.query(species == ('setosa', 'versicolor'))
y_0 = pd.Categorical(df['species']).codes
x_n = 'sepal_length'
x_0 = df[x_n].values
```

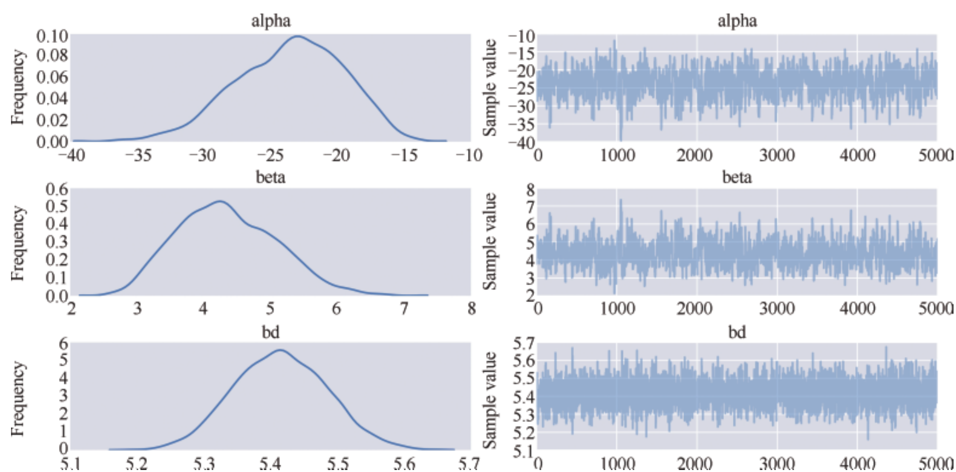
现在数据已经表示成了合适的格式，终于可以用 PyMC3 来建模了。留意下面代码中的第一部分与线性回归模型的相似之处。此外还留意两个确定变量：`theta` 和 `bd`。`theta` 是对变量 `mu` 应用逻辑函数之后的值，`bd` 是一个有边界的值，用于确定分类结果，稍后会详细讨论。还有一点值得提的是除了像下面这样明确写出逻辑函数的完整形式之外，还可以简单地使用 PyMC3 中的 `pm.math.sigmoid` 函数，该函数是 Theano 中 `sigmoid` 函数的一个别名。

对于其他线性模型而言，将数据进行中心化和（或）标准化处理会有利于采样。不过，在这个例子中，我们暂且不对数据做进一步处理。

```
with pm.Model() as model_0:
    alpha = pm.Normal('alpha', mu=0, sd=10)
    beta = pm.Normal('beta', mu=0, sd=10)

    mu = alpha + pm.dot(x_0, beta)
    theta = pm.Deterministic('theta', 1 / (1 + pm.math.exp(-mu)))
    bd = pm.Deterministic('bd', -alpha/beta)

    yl = pm.Bernoulli('yl', theta, observed=y_0)
    start = pm.find_MAP()
    step = pm.NUTS()
    trace_0 = pm.sample(5000, step, start)
chain_0 = trace_0[1000:]
varnames = ['alpha', 'beta', 'bd']
pm.traceplot(chain_0, varnames)
```



和往常一样，我们将后验的总结打印出来，然后将 bd 与通过另外一种方式得到的值进行比较。

```
pm.df_summary(trace_0, varnames)
```

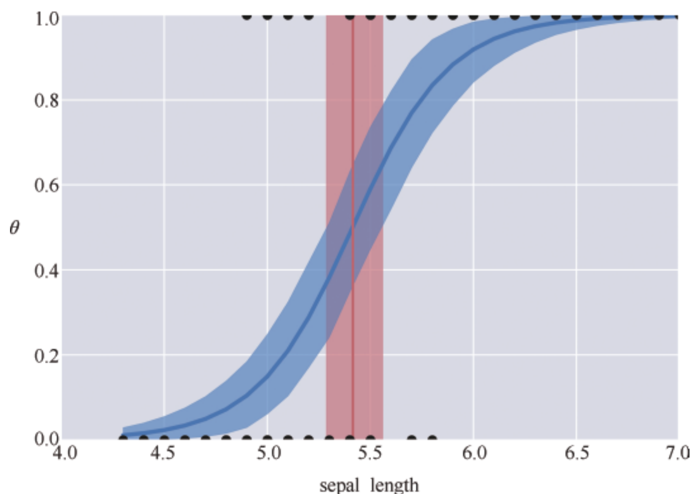
	mean	sd	mc_error	hpd_2.5	hpd_97.5
alpha	-23.49	4.07	1.77e-01	-31.38	-15.72
beta	4.34	0.75	3.28e-02	2.88	5.75
bd	5.42	0.07	1.09e-03	5.27	5.55

现在，我们将数据及拟合的 sigmoid（S- 型）曲线画出来。

```
theta = trace_0['theta'].mean(axis=0)
idx = np.argsort(x_0)
plt.plot(x_0[idx], theta[idx], color='b', lw=3);
plt.axvline(trace_0['bd'].mean(), ymax=1, color=''r'')
bd_hpd = pm.hpd(trace_0['bd'])
plt.fill_betweenx([0, 1], bd_hpd[0], bd_hpd[1], color='r', alpha=0.5)

plt.plot(x_0, y_0, 'o', color='k')
theta_hpd = pm.hpd(trace_0['theta'])[idx]
plt.fill_between(x_0[idx], theta_hpd[:,0], theta_hpd[:,1], color='b', alpha=0.5)

plt.xlabel(x_n, fontsize=16)
plt.ylabel(r'$\theta$', rotation=0, fontsize=16)
```



前面这张图表示了花萼长度与花的种类（setosa = 0, versicolor =1）之间的关系。蓝色的 S 型曲线表示 theta 的均值，这条线可以解释为：在知道花萼长度的情况下花的种类是 versicolor 的概率，即 $p(y=1/x)$。半透明的蓝色区间是 95%HPD 区间。注意：在这种情况下，逻辑回归其实就是回归，因为我们做的就是在给定特征（或者特征的线性组合）的条件下，回归出一个数据点的类别为 1 的概率。不过需要时刻注意观察到的变量是二值变量而推断的是连续的概率，因此我们需要引入一个规则将连续的概率转换成一个 0-1 结果。决策边界在图中用红色来表示，此外还有一个 95%HPD 区间。根据决策边界，x 值位于其左侧的（这里对应花萼长度）属于类别 0（setosa），而位于其右侧的属于类别 1（versicolor）。决策边界在 x 轴上的取值为对应 y=0.5 时的点，可以证明其结果是$-\dfrac{\alpha}{\beta}$，推导过程如下。

根据模型的定义，我们有如下关系：

$$\theta = logistic\ (\alpha + \beta \boldsymbol{X})$$

根据逻辑函数的定义，当 $\theta = 0.5$ 时，对应的输入为 0，则有：

$$0.5 = logistic\ (\alpha + \beta x_i) \Leftrightarrow 0 = \alpha + \beta x_i$$

移项后可以得出，当 $\theta = 0.5$ 时，对应有：

$$x_i = -\frac{\alpha}{\beta}$$

值得一提的是：决策边界是一个标量，也就是说是一个数值，这一点对于一维数据来说是合理的。因此我们只需要一个标量就可以将数据分成两组（或者

两类）。很重要的一点是：尽管决策边界看起来很合理，但是这里选取的 0.5 并不是什么特殊值，你完全可以选其他 0 到 1 之间的值。只有当我们认为将标签 0（setosa）错误地标为标签 1（versicolor）时的代价，与反过来将标签 1（versicolor）错误地标为标签 0（setosa）时的代价相同时，选取 0.5 作为决策边界才是可行的。不过大多数情况下，分类出错的代价并不是对称的。

做预测

一旦有了 α 和 β 之后，我们就能用其对新的数据进行分类。可以创建一个非常基础的函数，接受花萼长度作为参数，返回分类结果为 versicolor 的概率值。具体代码如下所示：

```
def classify(n, threshold):
    """
    A simple classifying function
    """
    n = np.array(n)
    mu = trace_0['alpha'].mean() + trace_0['beta'].mean() * n
    prob = 1 / (1 + np.exp(-mu))
    return prob, prob > threshold
classify([5, 5.5, 6], 0.4)
```

从前面的例子可以看出，很难只根据花萼长度就将 setosa 和 versicolor 这两类花区分出来。事实上如果你仔细看过 `joinplot` 画的图，就可以很清楚地观察到这点。仔细观察数据可以发现，类别为 versicolor 的花的花萼长度最小值约为 4.9，而类别为 setosa 的花的花萼长度最大值约为 5.8。换句话说，两种花的花萼长度在 4.9 到 5.8 的范围内有重叠。

如果只使用另外一种变量呢？请查看本章练习中的第一题，我们将在那里探讨这个问题。

5.2 多元逻辑回归

与多变量线性回归类似，多变量逻辑回归使用了多个自变量。这里将花萼长度与花萼宽度结合在一起，注意这里需要对数据做一些预处理。

```
df = iris.query(species == ('setosa', 'versicolor'))
```

```
y_1 = pd.Categorical(df['species']).codes
x_n = ['sepal_length', 'sepal_width']
x_1 = df[x_n].values
```

5.2.1　决策边界

如果你对如何推导决策边界不感兴趣的话，可以略过这个部分直接跳到模型实现部分。 根据模型，我们有：

$$\theta = logistic\ (\alpha + \beta_0 x_0 + \beta_1 x_1)$$

根据逻辑函数的定义，当逻辑回归的参数为 0 时，我们有 $\theta = 0.5$，也就是说：

$$0.5 = logistic\ (\alpha + \beta_0 x_0 + \beta_1 x_1) \Leftrightarrow 0 = \alpha + \beta_0 x_0 + \beta_1 x_1$$

移项之后可以得出，当 $\theta = 0.5$ 时，对于 x_1 我们有：

$$x_1 = -\frac{\alpha}{\beta_1} + \left(-\frac{\beta_0}{\beta_1} x_0\right)$$

这个决策边界的表达式与直线的表达式在数学形式上是一样的，其中第 1 项表示截距，第 2 项表示斜率，这里的括号只是为了表达上更清晰，如果你愿意的话完全可以去掉。为什么决策边界是直线呢？想想看，如果我们有一个特征，还有一维的数据，可以用一个点将数据分成两组；如果有两个特征，也就有一个 2 维的数据空间，从而我们可以用一条直线来对其分割；对于 3 维的情况，边界是一个平面，对于更高的维度，我们对应有一个超平面。事实上，从概念上讲，超平面可以大致定义为 n 维空间中 n-1 维的子空间，因此我们总是可以将决策边界称为超平面。

5.2.2　模型实现

如果要用 PyMC3 写出多元逻辑回归模型，可以借助其向量化表示的优势，只需要对前面的单参数逻辑回归模型做一些简单的修改即可。

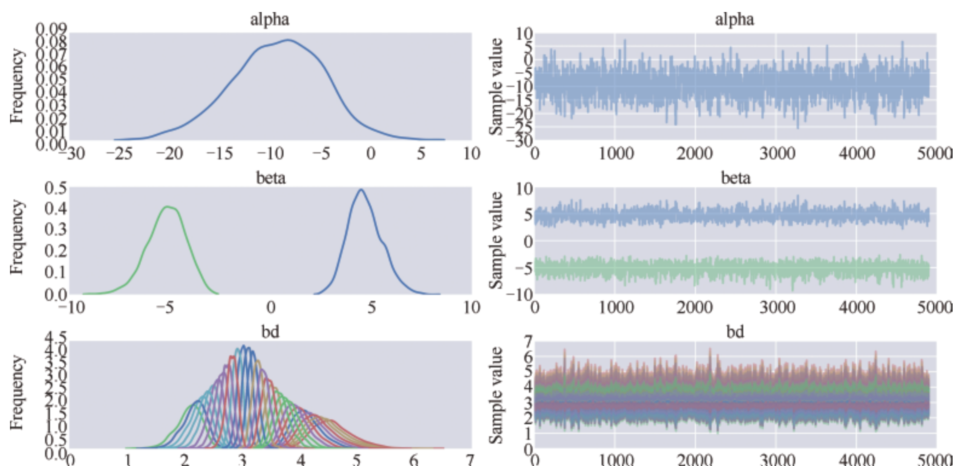
```
with pm.Model() as model_1:
    alpha = pm.Normal('alpha', mu=0, sd=10)
    beta = pm.Normal('beta', mu=0, sd=2, shape=len(x_n))

    mu = alpha + pm.math.dot(x_1, beta)
    theta = 1 / (1 + pm.math.exp(-mu))
```

```
    bd = pm.Deterministic('bd', -alpha/beta[1] - beta[0]/beta[1] * x_1[:,0])

    yl = pm.Bernoulli('yl', p=theta, observed=y_1)

    trace_1 = pm.sample(5000)
chain_1 = trace_1[100:]
varnames = ['alpha', 'beta']
pm.traceplot(chain_1)
```



注意在前面的图中，与前一个例子的区别是我们得到的不再是一个单独的决策边界的曲线，而是得到了 100 个，每个曲线对应一个数据点。

正如我们对单个预测变量所做的那样，可以将数据以及决策边界画出来，下面的代码中没有画 sigmoid（现在是一个 2D 的曲面）。如果愿意的话，你可以画出 3D 的图来表示 sigmoid 曲面。

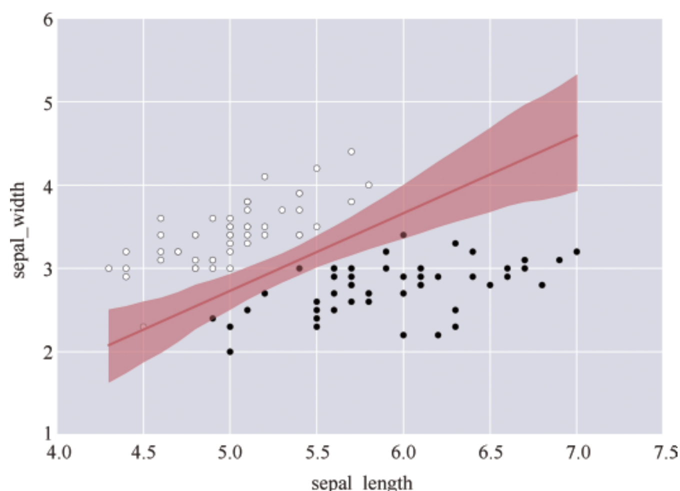
```
idx = np.argsort(x_1[:,0])
bd = chain_1['bd'].mean(0)[idx]
plt.scatter(x_1[:,0], x_1[:,1], c=y_0)
plt.plot(x_1[:,0][idx], bd, color='r');

bd_hpd = pm.hpd(chain_1['bd'])[idx]
plt.fill_between(x_1[:,0][idx], bd_hpd[:,0], bd_hpd[:,1], color='r', alpha=0.5);

plt.xlabel(x_n[0], fontsize=16)
plt.ylabel(x_n[1], fontsize=16)
```

我们已经看到过的，决策边界现在是一条直线，不要被 95%HPD 区间的曲线给误导了。图中半透明的曲线是由于在中间部分（也就是 HPD 区间较窄的部

分）有多条直线造成的。



5.2.3　处理相关变量

前一章中我们了解了在处理高度相关的变量时可能会遇到一些奇怪的问题。例如，如果用花瓣长度和花瓣宽度作为特征重跑前面的模型，会得到什么样的结果呢？

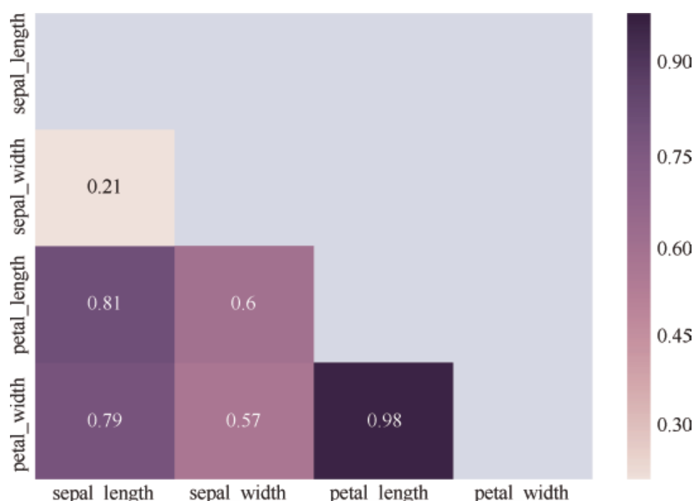
如果完成了上面的练习，你可能会注意到 beta 系数要比以前分布得更广，而且 95%HPD 区间（红色的区间）也更宽。下面的热力图显示了 4 个变量之间的相关性，可以看到（前一个例子中用到的）花萼长度与花萼宽度之间的相关性要远低于（后一个例子中用到的）花瓣长度与花瓣宽度之间的相关性。我们知道，相关的变量会得到更广的系数组合，因而能更好地解释数据，或者从另外一个角度来说，相关的变量限制模型的能力更弱。

当不同类别的数据能被完美区分时（即用线性模型分隔之后不同类的数据没有重叠），类似的问题就会出现。一个解决办法是不使用相关的变量，不过这个办法可能有时候并不适用。另外一个办法是给先验增加更多的信息，如果我们掌握了一些有用的信息，可以使用一些携带信息的先验，或者更一般地，可以使用一些弱信息的先验。Andrew Gelman 和 Stan 的开发团队建议在进行逻辑回归时使用如下先验：

$$\beta \sim Student\ t(0, v, s)$$

这里 s 的取值可以根据期望的尺度引入弱信息，正态参数 v 的值为 3 到 7 附近。该先验的含义是：我们期望系数比较小，同时引入了重尾，从而得到一个比高斯分布更鲁棒的模型。如果你忘记了的话，可以回忆一下第 3 章和第 4 章中的内容。

```
corr = iris[iris['species'] != 'virginica'].corr()
mask = np.tri(*corr.shape).T
sns.heatmap(corr.abs(), mask=mask, annot=True)
```



前面的图中，使用了一个掩码操作去掉了热力图中的上三角和对角线上的元素，因为这部分提供的是无效信息或者冗余信息。同时还需要注意，这里打印的是相关性的绝对值，原因是这里我们只关心相关性的强弱而不关心是正相关还是负相关。

5.2.4 处理类别不平衡数据

鸢尾花数据集的一个优点是：其中不同类别的样本量是均衡的，setosas、versicolors 和 virginicas 各有 50 个。鸢尾花数据集的流行归功于 Fisher，当然，Fisher 还让另一个东西也流行了——p 值。实际使用中许多数据集中的类别都是不平衡的，也就是说，其中一类数据的数量要远多于其他类别的数据。当这种情况发生时，逻辑回归会遇到一些问题，相比数据平衡时的情况，逻辑回归得到的边

界没有那么准确了。

现在我们看个实际的例子，这里我们随机从 setosa 类别中去掉一些数据点。

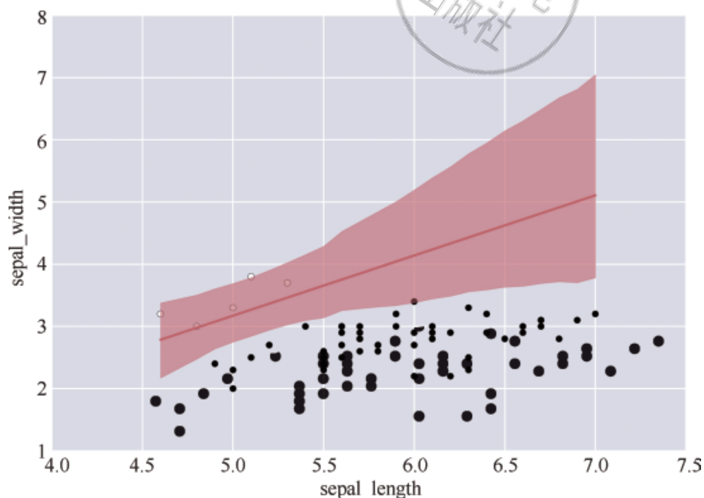
```
df = iris.query(species == ('setosa', 'versicolor'))
df = df[45:]
y_3 = pd.Categorical(df['species']).codes
x_n = ['sepal_length', 'sepal_width']
x_3 = df[x_n].values
```

和之前一样，你可以在自己电脑上运行多元逻辑回归，这里我直接将结果表示出来。

```
idx = np.argsort(x_3[:,0])
bd = trace_3['bd'].mean(0)[idx]
plt.scatter(x_3[:,0], x_3[:,1], c=y_3)
plt.plot(x_3[:,0][idx], bd, color='r');

bd_hpd = pm.hpd(trace_3['bd'])[idx]
plt.fill_between(x_3[:,0][idx], bd_hpd[:,0], bd_hpd[:,1], color='r', alpha=0.5);

plt.xlabel(x_n[0], fontsize=16)
plt.ylabel(x_n[1], fontsize=16)
```

可以看到，决策边界向样本量更少的类别偏移了，而且不确定性也比以前更大了。这是逻辑回归在处理不均衡数据时的常见表现。在一些数据中，类别之间的间隔可能不像这个例子中这么完美，此时用逻辑回归分类得到的结果中类别重叠的现象更严重。不过你可能觉得不确定性变得更大有可能是因为数据总量变少了，而不

只是因为 setosas 类别的数据相比 versicolors 更少。这是有可能的，你可以完成练习部分的第 2 题之后，亲自验证为什么不确定性变大的原因是数据不平衡。

5.2.5 如何解决类别不平衡的问题

一个显而易见的解决方案是，对数据集中的每一类都获取几乎相同数量的样本，如果你自己收集或者生成数据的话一定要记住这点。如果你并不能控制数据集，那么在对类别不平衡的数据进行解释时可要当心了，你可以通过检查模型的不确定性以及运行后验预测检查来确定模型是否对你有用。另外一种做法是给数据加入更多的先验信息，如果可能的话，可以运行本章剩余部分提到的一些其他模型。

5.2.6 解释逻辑回归的系数

解释逻辑回归的系数时一定要非常小心，因为逆连结函数（逻辑函数）引入了非线性。

$$\theta = logistic(\alpha + \beta \boldsymbol{X})$$

逻辑函数的逆函数是 logit 函数，其形式为：

$$logit(z) = \log\left(\frac{z}{1-z}\right)$$

因此，根据第一个等式，我们有：

$$logit(\theta) = \alpha + \beta \boldsymbol{X}$$

即

$$\log\left(\frac{\theta}{1-\theta}\right) = \alpha + \beta \boldsymbol{X}$$

注意这里模型中的θ是 y=1 的概率值，因此：

$$\log\left(\frac{p(y=1)}{1-p(y=1)}\right) = \alpha + \beta \boldsymbol{X}$$

其中$\frac{p(y=1)}{1-p(y=1)}$称作**发生比**，是另一种表示概率的方式。比如，掷骰子得到

点数 2 的概率为 1/6，因而其发生比为 1:5（即 0.2），也就是说有 1 个期望发生的事件和 5 个不期望发生的事件。

回到逻辑回归上，系数 β 的意义是：当 x 增加单位量的时候，发生比的对数增量。需要注意的是，β 并不是指当 x 增加时 $p(y=1)$ 的增量，因为二者之间的关系并不是线性的。如果 β 是正数，那么增加 x 会在某种程度上增加 $p(y=1)$，但是具体的增量取决于当前 x 的值。这一点在前面画的 S 型曲线上可以看出来，y 对 x 的斜率随着 x 的变化而变化，但是发生比的对数相对于 x 的斜率则是线性的。

5.2.7　广义线性模型

现在对本章所学的内容总结下，分析一下本章内容是如何与前一章中线性回归模型联系起来的。我们所做的就是将模型扩展应用到类别变量，具体做法是引入了逆连结函数，并且将高斯分布替换成了另外一种分布（伯努利分布）。总的来说就是：通过改变似然、先验及逆连结函数，我们把前一章中的线性回归模型应用到了不同的数据 / 问题上。

逻辑回归模型并非线性回归模型的唯一扩展。事实上，有一系列模型都可以看作是线性回归模型的一般形式，通常称为广义线性模型。统计学中一些常用的广义线性模型有：

- softmax 回归（下一章会见到），将逻辑回归应用到多于两个类别的分类问题。
- **方差分析**（ANalysis Of VAriance，ANOVA），其中有一个连续的因量和两个以上的离散的自变量。ANOVA 模型主要用来比较不同组之间的相似程度，该方法用的是线性回归模型。
- 泊松回归，我们将在第 7 章学习泊松回归模型的一个变种。

本书没有包含广义线性模型这部分内容，如果你想深入学习，特别是 ANOVA 模型，我强烈推荐 John Kruschke 的《Doing Bayesian Data Analysis》一书，书中针对如何构建基于广义线性模型的贝叶斯模型有很详细的介绍。

5.2.8 Softmax 回归或多项逻辑回归

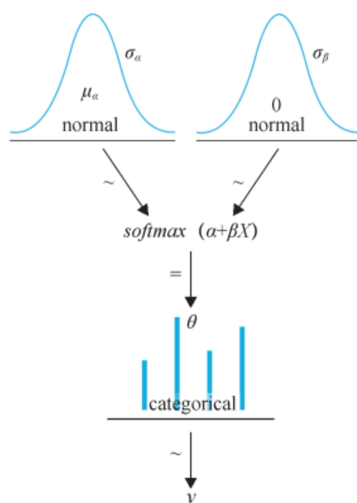
现在我们知道了如何处理二分类问题，接下来将我们所学的内容推广到多分类问题。一种做法是使用多项逻辑回归，该模型也被称作 softmax 回归，原因是这里使用的是 softmax 函数而非逻辑函数，softmax 函数的形式如下：

$$softmax_i(\boldsymbol{\mu}) = \frac{\exp(u_i)}{\sum \exp(u_k)}$$

要计算向量 $\boldsymbol{\mu}$ 中第 i 个元素对应的 softmax 输出，需要将该元素的指数除以向量 $\boldsymbol{\mu}$ 中每个元素的指数之和。softmax 函数保证了输出值为正数而且和为 1。当 k=2 时，softmax 函数就变成了逻辑函数。另外，softmax 函数与统计学中的玻尔兹曼分布形式是一样的，玻尔兹曼分布也是物理学中用来描述分子系统中概率分布的一个强大分支。

在玻尔兹曼分布中（某些领域中的 softmax）有一个称为温度的参数（T），在数学形式上前面式子中的 $\boldsymbol{\mu}$ 变成了 $\boldsymbol{\mu}/T$，当 $T \to \infty$时，概率分布变得非常均匀，因而所有状态都是等可能的；当 $T \to 0$ 时，只有最可能的状态会输出，因而 softmax 表现得就像一个 max 函数，这也是其名字来源。

softmax 回归模型与逻辑回归模型的另一个区别是：伯努利分布换成了类别分布。类别分布其实是伯努利分布推广到两个以上输出时的一般形式。此外，伯努利分布（抛一次硬币）是二项分布（抛多次硬币）的特殊情况，类似地，类别分布（掷一次骰子）是多项分布（掷 N 次骰子）的特殊情况。



这里继续使用鸢尾花数据集，不过这次用到其中的 3 个类别标签（setosa、versicolor 及 virginica）和 4 个特征（花萼长度、花萼宽度、花瓣长度及花瓣宽度），同时对数据进行标准化处理（也还可以做中心化处理），这样采样效率更高。

```
iris = sns.load_dataset('iris')
y_s = pd.Categorical(iris['species']).codes
x_n = iris.columns[:-1]
x_s = iris[x_n].values
x_s = (x_s - x_s.mean(axis=0))/x_s.std(axis=0)
```

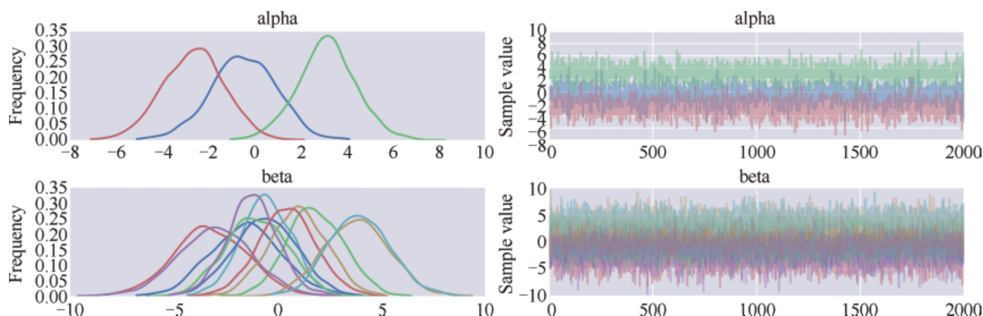
从 PyMC3 的代码可以看出，逻辑回归模型与 softmax 模型之间的变化很小，留意 alpha 系数和 beta 系数的长度。这段代码中用到了 Theano 中的 softmax 函数，根据 PyMC3 开发者的惯例，按 `import theano.tensor as tt` 这种方式导入的 Theano。

```
with pm.Model() as model_s:
    alpha = pm.Normal('alpha', mu=0, sd=2, shape=3)
    beta = pm.Normal('beta', mu=0, sd=2, shape=(4,3))

    mu = alpha + pm.dot(x_s, beta)

    theta = tt.nnet.softmax(mu)

    yl = pm.Categorical('yl', p=theta, observed=y_s)
    start = pm.find_MAP()
    step = pm.NUTS()
    trace_s = pm.sample(2000, step, start)
pm.traceplot(trace_s)
```

那么我们的模型表现如何呢？可以根据准确预测的样本个数来判断。下面的代码中使用了参数的均值来计算每个点分别属于 3 个类别的概率值，然

后使用 argmax 函数求出概率最大的类别作为结果，最后将结果与观测值进行比较。

```
    data_pred = trace_s['alpha'].mean(axis=0) + np.dot(x_s, trace_s['beta'].
mean(axis=0))
    y_pred = []
    for point in data_pred:
        y_pred.append(np.exp(point)/np.sum(np.exp(point), axis=0))
    np.sum(y_s == np.argmax(y_pred, axis=1))/len(y_s)
```

分类结果显示准确率约为 98%，也就是说，只错分了 3 个样本。不过，真正要评估模型的效果需要使用模型没有见过的数据，否则，可能会高估了模型对其他数据的泛化能力。下一章我们会详细讨论这个主题，目前暂且把它当做自动一致性检查，证明我们的模型运行正常。

也许你已经注意到了，后验（或者更准确地说，每个参数的边缘分布）看起来分布得很宽；事实上，它们与先验分布得一样宽。尽管我们能做出正确的预测，但这看起来并不令人满意。在前面的线性 / 逻辑回归问题中，对于有相关性的数据或者可以完美分割的数据，我们也遇到过类似不可识别的问题。在这个例子中，后验分布较广是因为受到了所有概率之和为 1 的限制。在这种情况下，我们用到的参数个数比实际定义模型所需要的参数个数更多。简单来说就是，假如 10 个数的和为 1，你只需要知道 9 个数就可以了，剩下的 1 个数可以用 1 减去这 9 个数之和算出来。解决这个问题的办法是将额外的参数固定为某个值（比如 0）。下面的代码展示了如何用 PyMC3 来实现。

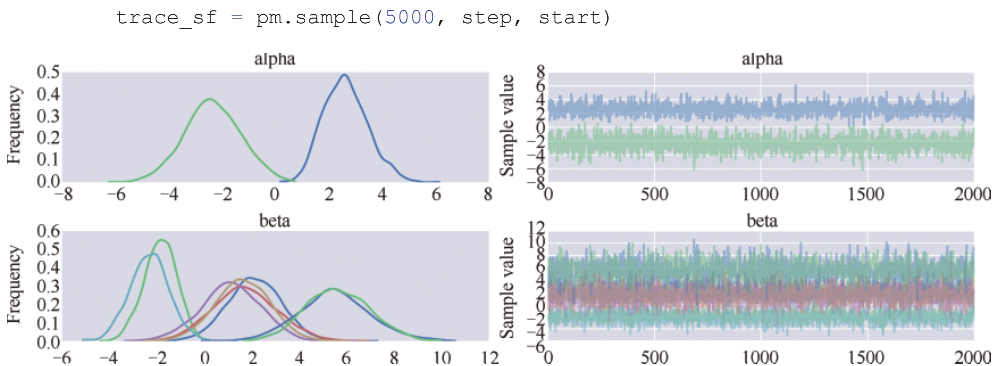
```
    with pm.Model() as model_sf:
        alpha = pm.Normal('alpha', mu=0, sd=2, shape=2)
        beta = pm.Normal('beta', mu=0, sd=2, shape=(4,2))

        alpha_f = tt.concatenate([[0] , alpha])
        beta_f = tt.concatenate([np.zeros((4,1)) , beta], axis=1)

        mu = alpha_f + pm.math.dot(x_s, beta_f)
        theta = tt.nnet.softmax(mu)

        yl = pm.Categorical('yl', p=theta, observed=y_s)
        start = pm.find_MAP()
        step = pm.NUTS()
```

```
    trace_sf = pm.sample(5000, step, start)
```



5.3　判别式和生成式模型

目前为止，我们已经讨论了逻辑回归及其扩展，所有这些情况都是直接计算 $p(y \mid x)$，也就是说，在知道 x 的条件下，计算出类别 y 的概率值。换句话说，我们所做的是直接根据自变量到因变量之间的关系进行建模，然后用一个阈值对得到的（连续的）概率值进行评判，从而得到分类结果。

上面这种方法不是唯一的，另一种方法是先对 $p(x \mid y)$ 建模，即类别计算特征的分布，然后再进行分类，这类模型称为生成式分类器，因为我们得到的模型可以从每个类别中生成采样。与此相反，逻辑回归属于判别式分类器，因为它只能判断一个样本是不是属于某一类别，并不能从每个类别中生成样本。

这里我们不打算深入生成式分类器模型，不过可以通过一个例子来说明这类模型用于分类的核心思想。我们只使用两个类别和一个特征，与本章的第一个例子用到的数据一样。

下面的代码用 PyMC3 实现了一个生成式分类器，从代码中可以看出，现在决策边界变成了高斯分布期望的估计值的均值，当分布是正态分布且标准差相同时，这个决策边界是正确的。这些假设是由一种称作**线性判别分析**（Linear Discriminant Analysis，LDA）的模型做出的，尽管名字上是判别式分析，不过 LDA 模型其实是生成式的。

```
with pm.Model() as lda:

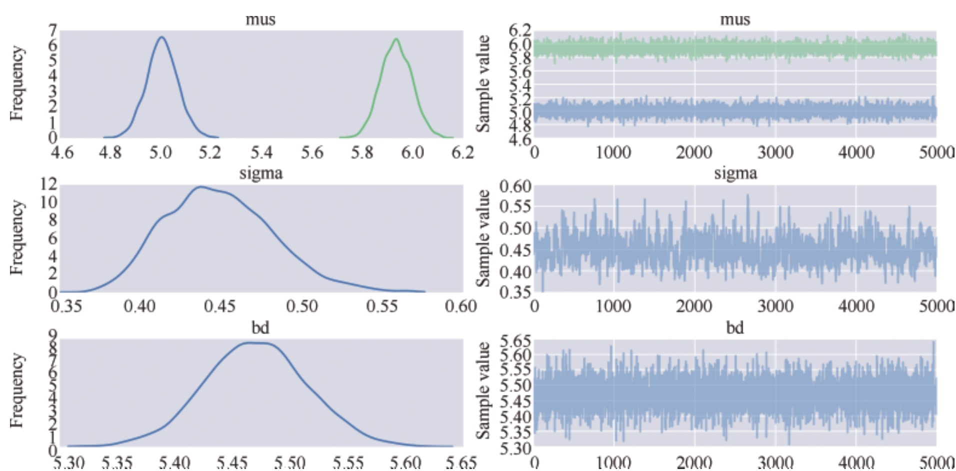
    mus = pm.Normal('mus', mu=0, sd=10, shape=2)
```

```
    sigmas = pm.Uniform('sigmas', 0, 10)

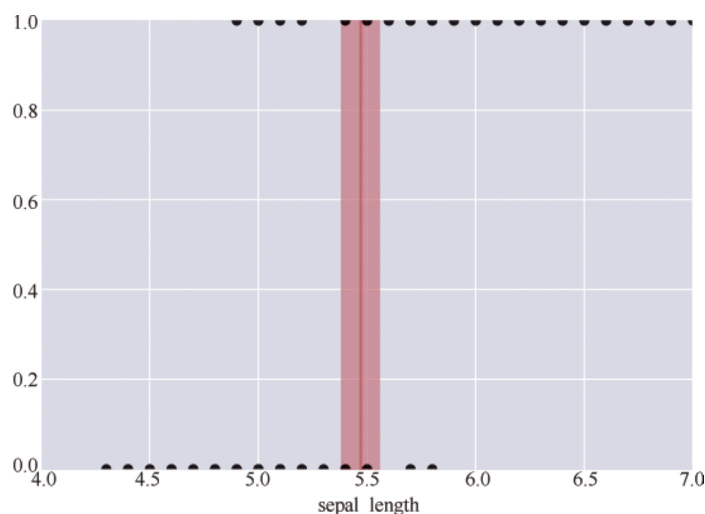
    setosa = pm.Normal('setosa', mu=mus[0], sd=sigmas[0], observed=x_0[:50])
    versicolor = pm.Normal('setosa', mu=mus[1], sd=sigmas[1], observed=x_0[50:])

    bd = pm.Deterministic('bd', (mus[0]+mus[1])/2)

    start = pm.find_MAP()
    step = pm.NUTS()
    trace = pm.sample(5000, step, start)
```



下面再将 setosa=0 和 versicolor=1 两个类别与花萼长度的关系画出来，一同画出来的还有一条红色的决策边界以及对应的 95%HPD 区间。



打印出模型的总结，对决策边界进行检查。

```
pm.df_summary(trace_lda)
```

	mean	sd	mc_error	hpd_2.5	hpd_97.5
mus__0	5.01	0.06	8.16e-04	4.88	5.13
mus__1	5.93	0.06	6.28e-04	5.81	6.06
sigma	0.45	0.03	1.52e-03	0.38	0.51
bd	5.47	0.05	5.36e-04	5.38	5.56

可以看到 LDA 模型得到了与逻辑回归类似的结果。

线性判别式模型可以使用多元高斯分布对类别建模，从而将其扩展到超过一个特征的情况。此外，还可以对不同类别的数据共享同一个方差（或者是同一个方差矩阵，对于超过一个特征的情况来说）的假设进行放松。这样便得到了称作**二次判别分析**（Quadratic Linear Discriminant，QDA）的模型，此时决策边界不再是线性的，而是二次的。

通常，当特征基本符合高斯分布时，LDA 或 QDA 的效果要比逻辑回归更好，如果假设不成立，逻辑回归的效果要更好一些。使用判别式模型分类的一个好处是：在模型中融合先验更容易（或者说更自然），比如我们可以将数据均值和方差的信息融入到模型中去。

需要注意：LDA 和 QDA 的决策边界是封闭式的，对于两个类别和一个特征的情况，应用 LDA 的时候只需要分别计算出每个类别分布的均值，然后求二者的均值就得到了决策边界。在前面的模型中，采用了更贝叶斯的一种方式；我们估计出两个高斯分布的参数，然后将这些估计融入公式中，不过公式是怎么来的呢？这里不深入更细节的内容，只需要知道要得到该公式，我们需要假设数据是符合高斯分布的，因此 LDA 只有在数据分布接近高斯分布的时候更有效。显然，在某些问题中，当我们想对正态性的假设放松的时候（比如 t 分布或者多元 t 分布等），就不能再使用 LDA（或 QDA）了，不过我们仍然可以用 PyMC3 从数值上计算出决策边界。

5.4　总结

本章我们学习了如何扩展单参数线性回归模型用于处理分类问题，在只有

两种类别时如何用逻辑回归进行贝叶斯分类，以及在超过两种类别时如何利用 softmax 回归进行贝叶斯分类。我们还学习了什么是逆连结函数，如何用来构建广义线性模型，广义模型大大地扩展了线性模型所能解决的问题。此外还学习了建模过程中可能需要注意的地方，比如遇到有关联的变量、完美分类的类别或者有偏的类别时应该怎么处理。我们重点关注的是判别式模型，稍微了解了下生成式模型，并学习了二者之间的主要区别。

5.5 深入阅读

- 《Doing Bayesian Data Analysis, Second Edition》的第 21 章和第 22 章。
- 《Statistical Rethinking》的第 10 章。
- 《An Introduction to Statistical Learning by Gareth James and others (second edition)》的第 4 章。
- 查看 PyMC3 中有关逻辑回归的例子：https://pymc-devs.github.io/pymc3/notebooks/GLM-logistic.html。这个例子还包含了我们下一章中将要讨论的模型比较相关的内容。

5.6 练习

（1）使用花瓣长度和花瓣宽度作为变量重跑第一个模型。二者的结果有何区别？两种情况下的 95%HPD 区间分别是多少？

（2）重跑练习（1），这次使用 t 分布作为弱先验信息。尝试使用不同的正态参数 v。

（3）回到第 1 个例子中，用逻辑回归根据花萼长度判断属于 setosa 还是 versicolor。尝试用第 1 章中的单参数回归模型来解决这个问题，线性回归的结果相比逻辑回归的效果如何？线性回归的结果能解释为概率吗？提示：检查 y 值是否位于 [0,1] 区间内。

（4）假设我们不用 softmax 回归，而是用单参数线性模型，并将类别编码为 setosa =0, versicolor =1, virginica = 2。在单参数线性回归模型中，如果我们交换类别的编码方式会发生什么？结果会保持一样还是会有所不同？

（5）在处理不均衡数据的例子中，将 `df = df[45:]` 改为 `df[22:78]`，这样做得到的数据点个数几乎没变，不过现在类别变得均衡了，试比较这两种情况的结果，哪种情况得到的结果与使用完整数据集得到的结果更相似呢？

（6）比较逻辑回归模型与 LDA 模型的似然，用函数 `sample_ppc` 生成预测数据并比较，确保理解其中的不同。



第 6 章 模型比较

所有模型都是错的，但某些是有用的。

——George Box

我们已经讨论过“所有模型都是错的”这一思想，原因是模型只是通过数据理解问题的一个近似工具，并非对真实世界的完整刻画。尽管每个模型都是错误的，但是并非每个模型都错得一样。某些模型在描述同一份数据时会错得更离谱。前一章中，我们重点关注了推断的问题，也就是根据数据推断参数的值。这一章，我们将关注另外一个问题：如何比较两个或多个模型。这个问题并不简单，而且目前也是数据分析中的一个核心问题。

本章将讨论以下内容：

- 奥卡姆剃刀、简洁性与准确率、过拟合与欠拟合；
- 正则先验；
- 信息量准则；
- 贝叶斯因子。

6.1 奥卡姆剃刀——简约性与准确性

假设对于同一个问题 / 数据有两个模型，二者对数据解释得同样好，应该选哪个模型呢？有一个准则叫做**奥卡姆剃刀**，描述的是如果对于同一现象有两种不同的假说，我们应该采取比较简单的那一种。关于奥卡姆剃刀的论证有很多，其中之一与波普尔的可证伪性标准有关，还有一种说法是从实用的角度提出的，因为更简单的模型相比复杂的模型更容易理解，另外还有一种论证是基于贝叶斯统计。这里暂且不深入这些论证的细节，只是将该准则当做一个有用

而合理的常识。

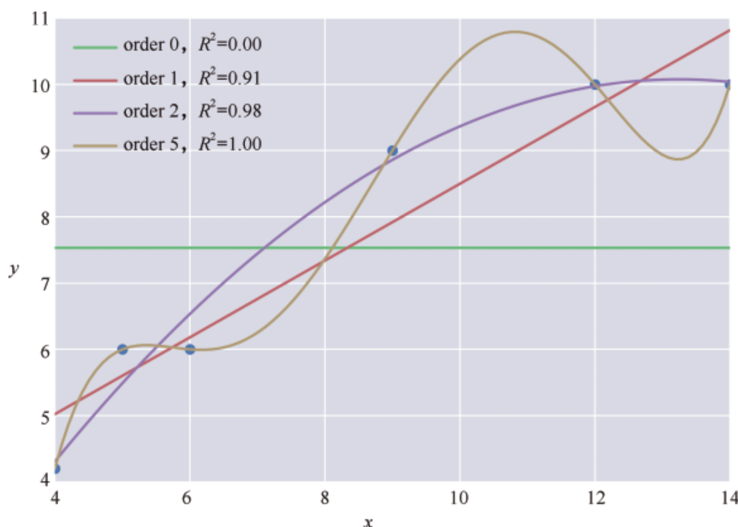
在比较模型时，通常还需要考虑模型的准确性，即模型对数据拟合得怎么样。我们已经见过了一些衡量准确性的标准，如测定 R^2 的系数，可以将其理解为线性回归中可解释方差的比例。如果我们有两个模型，其中一个模型对数据的解释比另一个更好，我们是否更倾向于该模型呢？换句话说，我们是否应该选准确率更高的模型呢？我们更倾向于更简单的模型。

直观上讲，在比较模型时，我们似乎更倾向于准确率较高，同时比较简单的模型。本章剩余部分将讨论如何平衡这两者。

这一章要比前面几章更偏理论一些（尽管我们还只是在讨论这个主题中很浅显的部分）。为了简化问题，这里引入一个例子来帮助理解如何平衡准确性与复杂性，实现从感性认识到理论证明的跨越。

在这个例子中，我们将使用一系列逐渐复杂的多项式来拟合一个非常简单的数据集，这里我们没有采用贝叶斯方法，而是采用最小二乘方近似来拟合线性模型。后者其实可以转化成一个带均匀先验的贝叶斯模型，因此，这里可以理解为还是用的贝叶斯模型，只不过我们走了个捷径。


```
x = np.array([4.,5.,6.,9.,12, 14.])
y = np.array([4.2, 6., 6., 9., 10, 10.])
order = [0, 1, 2, 5]
plt.plot(x, y, 'o')
for i in order:
  x_n = np.linspace(x.min(), x.max(), 100)
  coeffs = np.polyfit(x, y, deg=i)
  ffit = np.polyval(coeffs, x_n)
  p = np.poly1d(coeffs)
  yhat = p(x)
  ybar = np.mean(y)
  ssreg = np.sum((yhat-ybar)**2)
  sstot = np.sum((y - ybar)**2)
  r2 = ssreg / sstot
plt.plot(x_n, ffit, label='order {}, $R^2$= {:.2f}'.format(i,r2))
plt.legend(loc=2, fontsize=14)
plt.xlabel('$x$', fontsize=16)
plt.ylabel('$y$', fontsize=16, rotation=0)
```

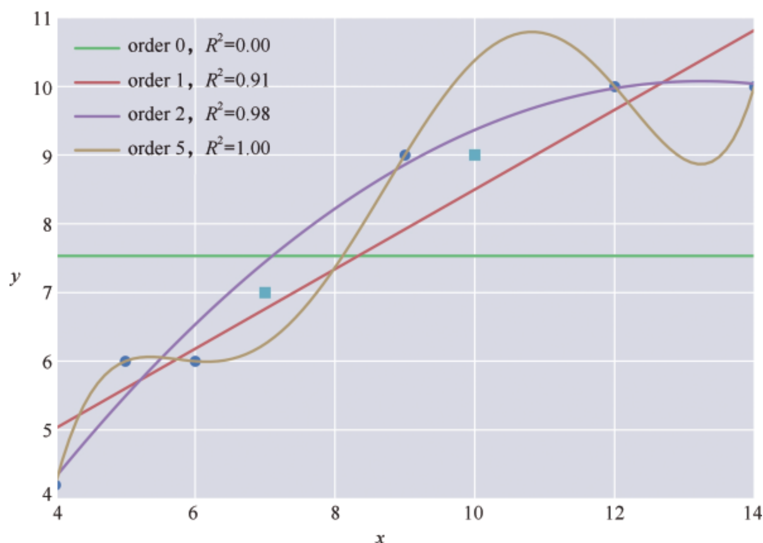


6.1.1 参数太多导致过拟合

从前面的图中可以看出，当模型的复杂度增加，对应的决定系数 R^2 也在上升。当多项式为 5 阶时，模型完美拟合了数据。前面章节中我们讨论过，通常，用多项式去解决实际问题并不是一个特别好的办法。

为什么 5 阶的多项式能够完美地拟合所有数据呢？原因是模型中参数的个数与样本个数相同，都是 6，因此，模型只是用另一种方式对数据进行了编码，此时模型并没有从数据中学到任何内容，只是记住了全部的数据。而且该模型对数据的预测看起来非常奇怪。从前面的图中可以看到，5 阶多项式对应的那条最佳拟合曲线一会儿高一会儿低，对比之下 1 阶和 2 阶的模型分别是直线和抛物线。直观上讲，直线或者抛物线要比一条弯弯曲曲的完美拟合所有数据的曲线要更简单而且更可信。从这个例子可以看出，准确率更高的模型可能并不是我们想要的。

下面的例子从另外一个重要的角度做出了解释。假设我们在原来数据集的基础上收集了更多的数据。比如，我们又收集到了 2 个点 [(10, 9), (7,7)]（如下面的图所示），此时 5 阶模型对比 1 阶或者 2 阶模型的效果如何呢？并不好。5 阶模型并没有从数据中学到任何有意义的模式，只是记住了所有见过的数据，因而对于潜在的未观测到的数据表现很差。



之所以泛化能力差是因为模型太灵活了，参数太多导致了过拟合。在统计学和机器学习中，**过拟合**是一个很常见的问题，一旦模型开始学习数据中模式之外的噪声时就会出现过拟合的问题，显然这里我们假设数据中首先是存在有意义的模式的。通常，一个模型的参数越多，适应数据的方式也就越多，因而更倾向于对数据过拟合。这一点是在使用复杂模型时的一个现实考虑，同时也是奥卡姆剃刀的一个现实佐证。

这个例子告诉我们，如果仅仅关注模型对数据的解释能力，很可能会被误导。原因是，（至少理论上）我们始终可以通过增加模型参数个数来提高准确率。这里引入一些词汇来让我们的讨论更清晰一些，我们称根据输入到模型的数据计算出来的准确率为**样本内准确率**，另外一种衡量模型有效性的方式是计算模型在没有见过的数据上的准确率，通常称为**样本外准确率**。

6.1.2　参数太少导致欠拟合

继续同样的例子，不过这次重点关注的不是非常复杂的模型，而是 0 阶的模型。在 0 阶模型中，所有的 beta 参数都为 0，因而两个变量之间的线性关系变成了只是描述因变量的一个高斯模型，注意对于 0 阶模型来说，自变量对模型不再有任何影响，而且模型只能捕捉到因变量的均值。换句话说，模型认为数据能够

通过因变量的均值以及一些高斯噪声来解释。我们称这种模型是欠拟合的，因为它实在太简单了，以至于并不能从数据中获取到有意义的模式。通常，一个参数很少的模型容易出现欠拟合。

6.1.3 简洁性与准确性之间的平衡

经常与奥卡姆剃刀准则一起提到的是爱因斯坦的一句名言“事情应该尽可能简单，但不必过于简单”。这就好像健康饮食，我们在建模的时候也需要保持某种平衡。理想状态下，我们希望模型既不过拟合又不欠拟合，因此，通常需要优化或者调整我们的模型来权衡二者，具体做法有很多种，比如，我们可以将对数据建模的目的看作是对数据的一种压缩后的表现；我们希望将数据尽可能简化，从而能够理解数据并做出预测。如果模型对数据表示压缩得非常严重，那么会丢失一些细节信息，因而可能只得到一些很简单的总结，比如均值；相反，则会得到太多噪声，极限情况下，我们得到的可能是没有任何压缩的数据的另一种表示。

过拟合与欠拟合之间的平衡可以从偏差扰动平衡的角度来讨论，我用一个例子来解释这个概念。假设我们有一个模型能够拟合数据集中的每个点，就像前面的 5 阶模型一样。设想一下，如果我们重新取 6 个数据点，然后调整模型以适应新的数据点，这样每次都得到一个新的曲线，如此重复多次。由于模型可以拟合每组数据中的细节，因而我们的预测结果会有很大的方差，我们称该模型的方差很大，相反，如果有一个受限的模型，比如一条直线，那么，它总是会试图容纳一条直线。一个**高偏差**的模型会有很大的偏见（如果用拟人的方式来描述的话），或者说惯性很大。

前面例子中 1 阶模型要比 2 阶模型的偏差更高而方差更低，后者会得到不同的曲线（直线是其中的一个特例），总结如下。

- **高偏差**是因为模型不能很好地适应数据。高偏差可能使得模型不能捕捉到数据中一些关键的模式，因而导致欠拟合。
- **高方差**是由于对数据中的细节很敏感。高偏差会导致模型捕捉到数据中的噪声，因而可能会导致过拟合。

总的来说，如果提升一个方面，就会导致另一方面下降，这也是为什么人们称之偏差 - 方差平衡，而我们最希望得到的是二者平衡的模型。

6.2　正则先验

使用（弱）信息先验是给模型引入偏差的一种方式，如果引入得合理是一件好事，因为这有利于避免过拟合。

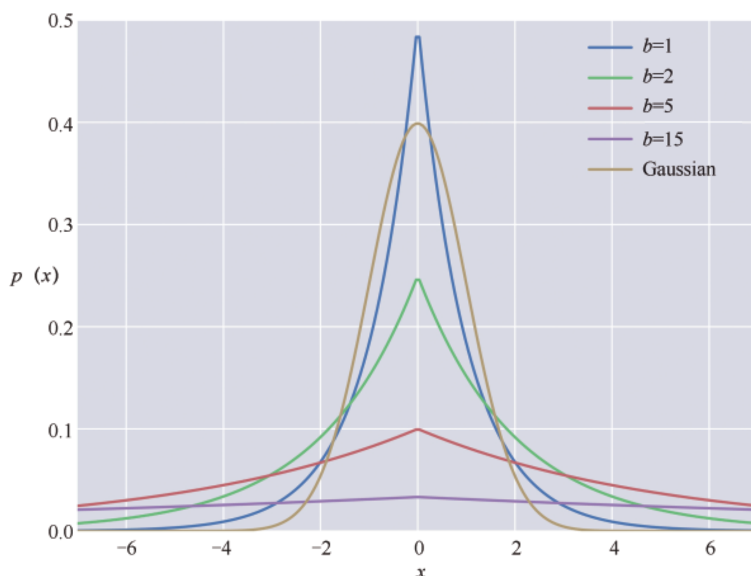
正则化的思想非常强大，在贝叶斯框架之外也有许多应用。在一些领域中，这种思想称作**吉洪诺夫正则化**。在非贝叶斯统计中，对于最小二乘法，正则化的思想有两种形式，分别是：岭回归和 Lasso 回归。从贝叶斯的角度来看，岭回归可以解释为对（线性回归中的）beta 系数使用了正态分布的先验，而且该正态分布的标准差很小（而不是习惯上的很大），因而达到把 beta 系数限制在 0 附近的目的，而 Lasso 回归则可以看作是对 beta 系数使用了拉普拉斯先验。标准形式的岭回归和 Lasso 回归对应的是点估计。如果严格应用贝叶斯分析的话是没法得到后验分布的。

在继续深入之前，我们先花点时间讨论下拉普拉斯分布。这个分布与高斯分布很像，不过它的 1 阶导数在 0 附近没有定义，因为该分布在 0 附近有一个很尖锐的峰值（具体可以看下图）。拉普拉斯分布相比高斯分布整体更靠近 0，因而当我们使用它作为先验的时候，会使得参数趋近于 0。也就是说，Lasso 可以用于正则化和变量筛选（将某些特征或变量从模型中去掉）。

下面的代码生成了 4 组中心为 0 的不同尺度的拉普拉斯分布，此外还有一个均值为零的标准差为 1 的高斯分布作为比较。

```
plt.figure(figsize=(8, 6))
x_values = np.linspace(-10, 10, 300)
for df in [1, 2, 5, 15]:
    distri = stats.laplace(scale=df)
    x_pdf = distri.pdf(x_values)
    plt.plot(x_values, x_pdf, label='$b$ = {}'.format(df))

x_pdf = stats.norm.pdf(x_values)
plt.plot(x_values, x_pdf, label='Gaussian')
plt.xlabel('x')
plt.ylabel('p(x)', rotation=0)
plt.legend(loc=0, fontsize=14)
plt.xlim(-7, 7);
plt.savefig('B04958_06_03.png', dpi=300, figsize=[5.5, 5.5])
```



非常值得注意的一点是，这种人们广泛接受的正则化的思想非常自然地融合进了贝叶斯体系中。有人甚至说，既然大家都认同正则化是一个相当棒的思想，那么可以说每个人都在某种程度上是贝叶斯的，尽管他们并没有意识到甚至拒绝这个标签。

6.2.1 正则先验和多层模型

与我们刚才讨论的相一致，多层模型也可以被认为是一种正则化的方法。也就是通过引入超先验，将多层模型看作是从数据中学习先验的一种方法。 所以，从某种意义上说，因为我们正在从数据中学习先验，所以我们正在进行正规化，并且让数据告诉我们正规化的强度。 这也许是对多层模型和收缩更深刻的理解。 你可以思考一下在第四章里当我们使用多层模型对单个数据点拟合一条直线时，如何从正则化概念的角度来解释。

6.3 衡量预测准确性

从前面的例子可以看出，0 阶的模型太过简单，而 5 阶的模型又太复杂，那剩下两个呢？如何进行区分？我们需要一个准则来同时考虑准确性和简洁性。有

两种方法可以只使用样本内数据来估计样本外数据的预测准确性。

- **交叉验证**：这是一种经验性的策略，将数据分成多个子集，然后轮流将其中一个子集作为测试集，将剩余的子集作为训练集进行评估。
- **信息准则**：一系列概念的总称，可以看作是对交叉检验的一种数学近似。

6.3.1　交叉验证

一般来说，模型在样本内的准确率要比样本外高。由于我们同时需要训练模型并测试模型的性能，一个简单的做法是将数据分成两部分：

- 用于训练模型的训练集；
- 用于测试模型表现的测试集。

如果有许多数据的话，交叉验证是一种很好的方法。比如，晶体学家已经使用这种方法求解并验证分子结构数十年了。不过如果数据不多的话，前面这种做法可能不合适，因为这会进一步减少训练模型和评估模型准确性的有限信息。

为了绕过缺少数据的问题，一个非常简单而且在大多数情况下都非常有效的做法是进行交叉验证。将数据分成 K 份，比如说 5 份，同时尽可能让每份数据相同（数据的个数以及数据的一些其他特征，比如每个类别的数量），然后使用其中的 $K-1$ 份数据用于训练模型（在这个例子中是 4 份），将剩下的一份数据用于验证模型。重复该过程 K 轮，每一轮都使用不同的数据用于验证，最后对每一轮的验证结果求均值。整个过程称作 **K- 折交叉验证**，其中 K 表示数据拆分成的份数，在这个例子中，我们称为 5 折交叉验证。当 K 与数据集中样本的个数相同时，我们称之为**留一交叉验证**（Leave-One-Out Cross-Validation，LOOCV）。有时候在做留一交叉验证时，验证的轮数可以小于样本总数。

交叉验证是一个非常简单而且强大的思想，不过对于某些模型或者某些量很大的数据而言，交叉验证的计算量可能超出我们能接受的范围。许多人尝试提出了一些其他更容易计算的量，来对交叉验证得到的结果进行近似，或者应用到不能直接使用交叉验证的情况，下一节将会详细介绍。

6.3.2 信息量准则

信息量准则是一系列用来比较模型对数据拟合程度的方法，这类方法引入了一个惩罚项来平衡模型的复杂度。换句话说，信息量准则形式化地表示了我们们在本章开头建立的一些直觉，用一种合适的方式平衡模型对数据的解释能力和模型的复杂程度。

这些衡量方式的推导过程与信息论相关，不过这超出了本书的范围，这里我们只从实用的角度去理解这些概念。

log似然与偏差

一种衡量模型对数据的拟合程度的方法是计算模型预测结果与真实数据之间的均方差。

$$\frac{1}{n}\sum(y_i - E(y_i \mid \theta))^2$$

其中，$E(y_i \mid \theta)$是根据预估参数值得到的预测值，可以看到基本上就是观察值和预测值之间的平均差异，求平方是为了保证正负误差不会相互抵消，此外相比一些其他的衡量指标（比如绝对误差），该衡量标准更强调较大的误差。

在数据已经是正态分布的情况下，前面的衡量方法计算简单而且很有用，不过更通用的一种方法是计算 log 似然：

$$\log\ p(y \mid \theta)$$

在一元线性回归模型中，log 似然与二次均方差是成比例的。

由于历史原因，在实践中人们通常不直接使用 log 似然，而是使用一个称作偏差平方和的量：

$$-2\log\ p(y \mid \theta)$$

偏差平方和在贝叶斯方法和非贝叶斯方法中类似，区别在于，贝叶斯框架中 θ 是从后验中估计出来的（和其他从后验中得到的量一样），是一个分布。相反，在非贝叶斯方法中，θ 是一个点估计。在使用偏差平方和的时候，需要注意以下

两点。

- 偏差平方和越小，log 似然的值越大，模型的预测结果与数据越吻合。因此我们希望偏差平方和越小越好。
- 偏差平方和衡量的是样本内的模型准确率，因而复杂的模型通常会比简单模型的偏差平方和小。因而需要给复杂模型加入惩罚项。

下面我们将学习几个不同的信息校准方法，它们的共同点是都使用了偏差平方和及正则项，区别在于偏差平方和和惩罚项的计算方式不同。

赤池信息量准则

赤池信息量准则（Akaike Information Criterion，AIC）是一个广泛应用的信息量准则，其定义如下：

$$AIC = -2\log\ p(y \mid \hat{\theta}_{mle}) + 2p_{aic}$$

其中，p_{aic}表示参数的个数，$\hat{\theta}_{mle}$是 θ 的最大似然估计。最大似然估计在非贝叶斯方法中经常用到，等价于使用贝叶斯方法中的均匀先验的最大后验估计。注意这里$\hat{\theta}_{mle}$是点估计而不是分布。 前面的表达式可以表示成如下形式：

$$AIC = -2(\log p(y \mid \hat{\theta}_{mle}) - p_{aic})$$

同样，这里的 −2 是出于历史原因。从实用的角度来看，上式中的第 1 项考虑的是模型对数据的拟合效果，第 2 项衡量的是模型复杂度。因此，如果两个模型对数据的解释能力相同，但是其中一个比另一个的参数更多的话，AIC 会告诉我们应该选择参数更少的那个。

AIC 对于非贝叶斯方法来说很有用，但是对于贝叶斯方法可能会有些问题。原因之一是 AIC 没有使用后验，因而将估计中的不确定信息丢失了，此外假设用到的是均匀先验，因而该准则对于使用非均匀先验的模型来说就不太合适了。在使用非均匀先验的时候，我们不能简单地计算模型中参数的个数。合理使用非均匀先验相当于对模型使用了正则，因而会降低过拟合的可能，也就是说带正则模型的有效参数个数比真实的参数个数要少。类似的情况在多层模型中同样会出现，毕竟多层模型可以看作是从数据中学习先验的有效方式。

偏差信息量准则

一种获取贝叶斯形式的 AIC 的方法是从后验中获取信息，同时从模型和数据中估计出参数的个数，这可以用**偏差信息量准则**（Deviance Information Criterion，DIC）来衡量：

$$-2 \times \log\ p(y \mid \hat{\theta}_{post}) + 2p_{dic}$$

可以看到 DIC 和 AIC 非常像，区别在于现在是从$\hat{\theta}_{post}$计算的偏差，即 θ 的后验均值，而且我们现在使用p_{dic}来代表模型的有效参数个数，用下式来表示：

$$\hat{D}(\theta) - D(\hat{\theta})$$

上式的含义是偏差的均值减去均值的偏差。如果得到的是一个有峰值的后验，θ 聚集在$\hat{\theta}$附近，那么上式中的左右两项会比较相似，因而p_{dic}会很小，相反，如果后验分布很广，那么会有更多的 θ 偏离 $\hat{\theta}$，因而$\hat{D}(\theta)$会很大，从而p_{dic}也就很大。

可以说，DIC 是更偏贝叶斯形式的 AIC，不过，DIC 没有使用完整的后验，而且对于弱先验来说，根据 DIC 计算得到的参数的有效个数会有些问题，为了解决这个问题，有一些替代方案。不过 DIC 对于这里的讨论来说足够了，PyMC3 中也已经实现了 DIC。

通用信息量准则

通用信息量准则（Widely Available Information Criterion，WAIC）与 DIC 类似，不过更偏贝叶斯，因为它使用的是整个后验分布。和 AIC、DIC 一样，我们可以看到 WAIC 同样有两项，一项衡量模型对数据的拟合效果；另外一项衡量模型的复杂程度。

$$WAIC = 2lppd + 2p_{waic}$$

这里 *lppd* 是点预测的 log，可以用下式近似：

$$lppd = \sum \log\left(\frac{1}{S}\sum p(y_i \mid \theta^s)\right)$$

这里首先从后验中采样 S 个样本并计算似然的均值，然后对于数据集中的 N 个数据点都重复该过程并求和。此外得到的有效参数个数可以用如下形式计算：

$$p_{waic} = \sum v_{s=1}^{s}(\log\ p(y_i \mid \theta^s))$$

也就是说，我们根据从后验中得到的 S 个采样值计算 log 似然的偏差，然后对 N 个数据点都重复该过程并求和。直观上看这似乎与 DIC 中计算有效参数个数的方法一样。在前面 AIC 部分讨论过，模型越灵活，就越倾向于得到分布更广（更分散）的后验。

帕累托平滑重要性采样与留一交叉验证

该方法用于近似 LOOCV 的结果，不过不需要真的计算 LOOCV。这里不过多深入其细节，其核心思想是通过对似然重采样去近似 LOOCV，可以通过一种重要性采样的技巧实现。该方法的问题是得到的结果不稳定，为了解决稳定性的问题，有人提出了一种新的方法叫做 PSIS，可以用于计算更可靠的 LOOCV 估计，其含义类似，值越小，模型得到的估计预测准确率越高。

贝叶斯信息准则

与逻辑回归的名字一样，这个名称同样有误导性。**贝叶斯信息准则**（Bayesian Information Criterion，BIC）是用来校正 AIC 的一些问题的，作者提出了一种贝叶斯校正方法。不过 BIC 实际上不是贝叶斯的方法，而是更像 AIC，因而假设先验是平坦的并使用最大似然估计。

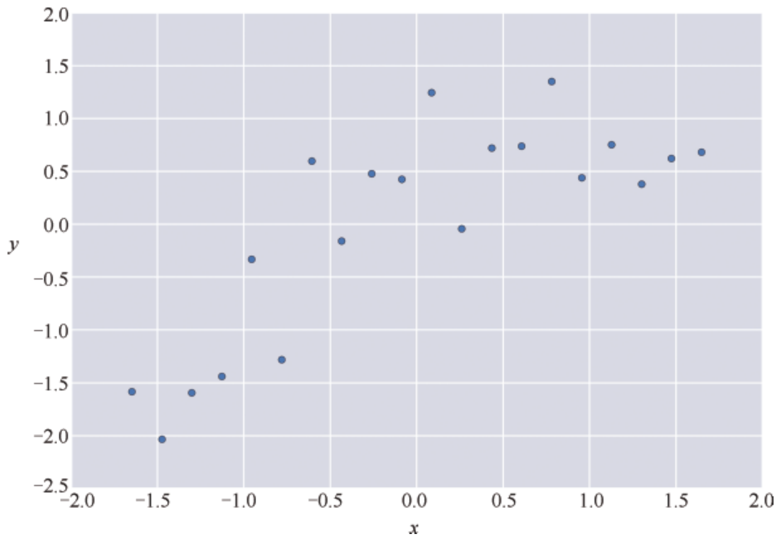
更重要的是，BIC 与我们见过的其他信息准则都不一样，它更像后面章节中会讨论到的贝叶斯因子。基于这些理由，同时参考了 Andrew Gelman 的建议，这里暂不深入讨论或使用 BIC。

6.3.3　用 PyMC3 计算信息量准则

通过 PyMC3 可以很容易计算出信息准则，只需要调用一个函数即可。为了简化它们的使用，我们将构建一个简单的模型，首先定义一些数据并标准化。

```
real_alpha = 4.25
real_beta = [8.7, -1.2]
```

```
data_size = 20
noise = np.random.normal(0, 2, size=data_size)
x_1 = np.linspace(0, 5, data_size)
y_1 = real_alpha + real_beta[0] * x_1 + real_beta[1] * x_1**2 + noise
order = 2
x_1p = np.vstack([x_1**i for i in range(1, order+1)])
x_1s = (x_1p - x_1p.mean(axis=1, keepdims=True))/x_1p.std(axis=1,
keepdims=True)
y_1s = (y_1 - y_1.mean())/y_1.std()
plt.scatter(x_1s[0], y_1s)
plt.xlabel('$x$', fontsize=14)
plt.ylabel('$y$', fontsize=14, rotation=0)
```



从图中可能看得不是特别明显，不过从代码中可以看到，我们得到的数据可以用二阶多项式去拟合。假设我们有理由认为线性回归是一个不错的模型，因此我们分别使用两个模型去拟合数据并用信息准则比较它们，先从线性回归模型开始。

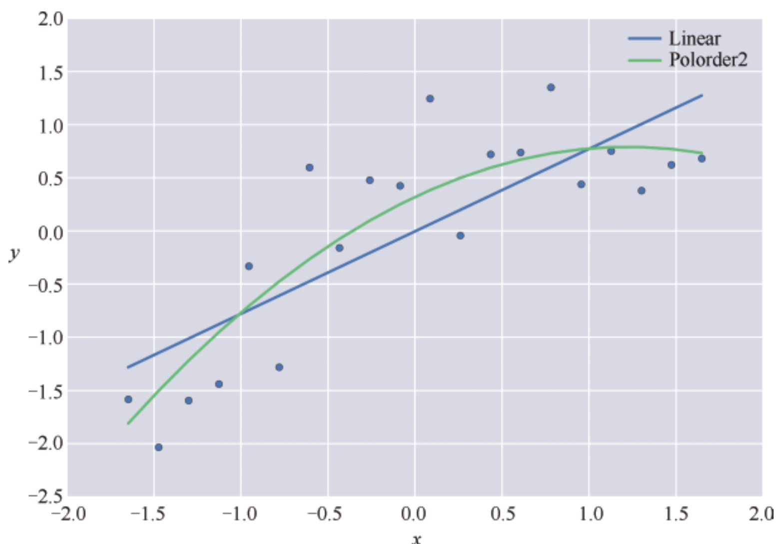
```
with pm.Model() as model_l:
    alpha = pm.Normal('alpha', mu=0, sd=10)
    beta = pm.Normal('beta', mu=0, sd=10)
    epsilon = pm.HalfCauchy('epsilon', 5)
    mu = alpha + beta * x_1s[0]
    y_pred = pm.Normal('y_pred', mu=mu, sd=epsilon, observed=y_1s)
    trace_l = pm.sample(2000)
chain_l = trace_l[100:]
```

为了节省版面，这里我们省略了 traceplot 以及一些其他画图测试的内容，不过你自己实现的时候可别省略了这部分内容，接下来继续用二阶模型去拟合。

```
with pm.Model() as model_p:
    alpha = pm.Normal('alpha', mu=0, sd=10)
    beta = pm.Normal('beta', mu=0, sd=10, shape=x_1s.shape[0])
    epsilon = pm.HalfCauchy('epsilon', 5)
    mu = alpha + pm.math.dot(beta, x_1s)
    y_pred = pm.Normal('y_pred', mu=mu, sd=epsilon, observed=y_1s)
    trace_p = pm.sample(1000)
chain_p = trace_p[100:]
```

现在我们将结果和最佳拟合直线画出来。

```
alpha_l_post = chain_l['alpha'].mean()
betas_l_post = chain_l['beta'].mean(axis=0)
idx = np.argsort(x_1s[0])
y_l_post = alpha_l_post + betas_l_post * x_1s[0]
plt.plot(x_1s[0][idx], y_l_post[idx], label='Linear')
alpha_p_post = chain_p['alpha'].mean()
betas_p_post = chain_p['beta'].mean(axis=0)
y_p_post = alpha_p_post + np.dot(betas_p_post, x_1s)
plt.plot(x_1s[0][idx], y_p_post[idx], label='Pol order {}'.format(order))
plt.scatter(x_1s[0], y_1s)
plt.legend()
```



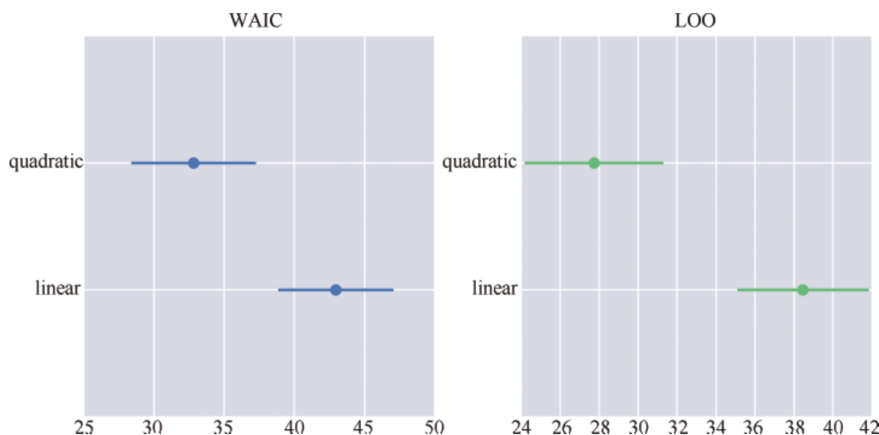
想要用 PyMC3 得到 DIC，我们需要将迹作为参数传给 `dic` 函数，如果调用的位置发生在 `with` 语句内，那么对应的模型参数会自动猜出来，当然也可以显示指定：

```
pm.dic(trace=trace_l, model=model_l)
```

同样，要计算 WAIC 调用 `pm.waic()` 即可，计算 LOO 则只需调用 `pm.loo()`。对于 WAIC 和 LOO，PyMC3 会返回一个点估计和相应的标准差，我们可以用标准差来评估 WAIC（或 LOO）估计的不确定性。不过需要注意的是，由于标准差的估计值假设是正态的，因此当样本量很小的时候不是很可靠。

```
plt.figure(figsize=(8, 4))
plt.subplot(121)
for idx, ic in enumerate((waic_l, waic_p)):
    plt.errorbar(ic[0], idx, xerr=ic[1], fmt='bo')
plt.title('WAIC')
plt.yticks([0, 1], ['linear', 'cuadratic'])
plt.ylim(-1, 2)

plt.subplot(122)
for idx, ic in enumerate((loo_l, loo_p)):
    plt.errorbar(ic[0], idx, xerr=ic[1], fmt='go')
plt.title('LOO')
plt.yticks([0, 1], ['linear', 'cuadratic'])
plt.ylim(-1, 2)
plt.tight_layout()
```



下面讨论一下关于 WAIC 和 LOO 计算结果的可靠性。

计算 WAIC 或 LOO 时，你可能会得到这样一条警告信息：二者的计算结果可能是不可靠的。该警告会根据一个经验性的值抛出（参考本章深入阅读部分）。尽管这不是多大的问题，不过它可能意味着计算过程中存在问题。WAIC 和 LOO 相对较新，我们可能还需要构建一些合适的方法得到它们的可靠性。不管如何，如果你遇到这种情况，首先确保你有足够多的样本，同时模型的迹有较好的混合度，然后是查看是否选用了一个“老化”的值。如果你还是得到警告，LOO 的作者建议使用更鲁棒的模型，比如用 t 分布替换高斯分布。如果这些建议都没有效果，那么你可能需要考虑一些其他方法，比如直接使用 K- 折交叉验证。

6.3.4　解释和使用信息校准

信息校准的一个简单应用是进行模型选择，直接选出信息准则值较小的模型并忽略其他模型即可，因而我们可以根据前面的那些图表得出以下结论：两种衡量标准都认为最好的模型是 2 阶的模型。

模型选择非常简单，但是这里我们抛掉了不确定性信息，这感觉就像是我们计算出了完整的后验分布却只保留了后验的均值，其后果是我们可能对得出的结论过于自信。

另一种做法是进行模型选择的同时，汇报和讨论不同模型的信息准则值以及后验预测检查的结果。将问题中的背景、数据和检查都分享出来很关键，因为这样人们才能更好地理解我们方法的极限和不足。如果你在学术界，你可以在论文、演示等讨论部分按照这种方式进行。

另外一种方法是进行模型平均，其思想是对每个模型加权之后生成一个元模型。计算权重的一种方式是：

$$w_i = \frac{\exp(-1/2dIC_i)}{\sum_j^M \exp(-1/2dIC_j)}$$

这里，dIC_i是第 i 个信息准则的值与最小值之间的差。

我们可以根据任意信息准则计算出一系列权重，不过我们显然不能将其混合在一起。上面的公式是一种启发式的方法，根据信息准则的值计算每个模型的相对概率，其中的分母仅仅是一个归一项，使得所有权重之和为 1。

还有一些其他方式对模型求平均，比如显式的构建一个模型将我们所有的模型包含在一起，然后进行参数推断，在后面贝叶斯因子部分，我们将讨论其中的一种形式。

除了对离散的模型求平均之外，有时候我们还可以将其看作是连续的。一个简单的例子就是，假设我们有一个抛硬币问题以及两个不同的模型，其中之一的先验偏向正面朝上，另一个偏向于反面朝上。我们可以分别用两个模型去拟合并用 dIC 权重求平均，除此之外还可以构建一个分层模型估计先验分布，注意这里构建的不再是两个离散的模型了，而是一个连续的模型，其中包含两个离散的模型作为特例。哪种方法更好呢？还是要具体问题具体分析，最终使用哪一个取决于实际问题是更适合用离散模型还是连续模型去描述。

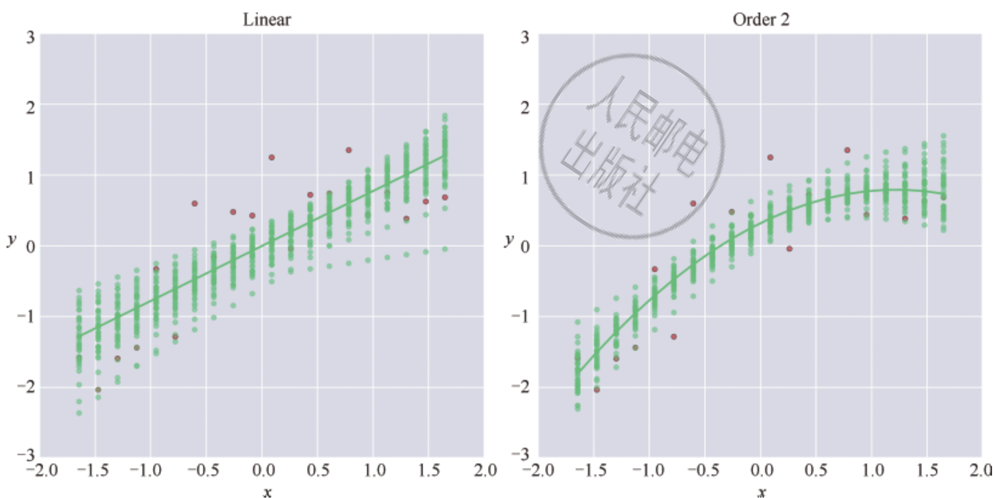
6.3.5 后验预测检查

前面几章中，我们介绍了后验预测检查的概念，并将其作为评估模型对数据的解释能力的一种方式，我们称之为完备性检查。进行后验预测检查的目的并不是为了说模型是错的，我们的目的是想知道哪部分数据拟合得不够好，从而知道模型的极限在哪或者如何优化模型。这里我们重新回顾这个话题，以此强调后验预测检查可以用来比较模型并理解模型之间的不同点。

```
plt.subplot(121)
plt.scatter(x_1s[0], y_1s, c='r');
plt.ylim(-3, 3)
plt.xlabel('x')
plt.ylabel('y', rotation=0)
plt.title('Linear')
for i in range(0, len(chain_l['alpha']), 50):
    plt.scatter(x_1s[0], chain_l['alpha'][i] + chain_l['beta'][i]*x_1s[0],
c='g', edgecolors='g', alpha=0.05);
plt.plot(x_1s[0], chain_l['alpha'].mean() + chain_l['beta'].
```

```
mean()*x_1s[0], c='g', alpha=1)

    plt.subplot(122)
    plt.scatter(x_1s[0], y_1s, c='r');
    plt.ylim(-3, 3)
    plt.xlabel('x')
    plt.ylabel('y', rotation=0)
    plt.title('Order {}'.format(order))
    for i in range(0, len(chain_p['alpha']), 50):
        plt.scatter(x_1s[0], chain_p['alpha'][i] + np.dot(chain_p['beta'][i],
x_1s), c='g', edgecolors='g', alpha=0.1)
    idx = np.argsort(x_1)
    plt.plot(x_1s[0][idx], alpha_p_post + np.dot(betas_p_post, x_1s)[idx],
c='g', alpha=1)
```



6.4　贝叶斯因子

在贝叶斯的世界中，评估和比较模型的另一种方式是使用贝叶斯因子。

使用贝叶斯因子时，在单个模型中可能存在某些先验对后验分布没有实际影响，但却对贝叶斯因子有较大影响。前面的例子中你可能注意到了，通常一个标

准差为 100 的正态先验与标准差为 1000 的正态先验对后验的效果差不多，而贝叶斯因子则会受到模型中这类变化的影响。贝叶斯因子的另外一个问题是计算起来可能要比推断过程更复杂。最后一点是，贝叶斯因子可以用来做假设检验，这本来不是什么问题，不过许多作者指出，类似本书提到的基于模型和推断的思维，要比基于假设检验的这类思维在大多数问题上更好。为了更好地理解什么是贝叶斯因子，这里我们再重新写一遍贝叶斯理论：

$$p(\theta \mid y) = \frac{p(y \mid \theta) p(\theta)}{p(y)}$$

其中，y 表示数据，θ 表示参数，你也可以写成如下形式：

$$p(\theta \mid y, M) = \frac{p(y \mid \theta, M) p(\theta \mid M)}{p(y \mid M)}$$

两个式子的唯一区别是：重写后的式子中显式地描述了推断过程中依赖的模型 M。其中分母称为证据或者边缘似然。目前为止，得益于推断引擎（如 Metropolis 和 NUTS），我们将这一项省略了。这里可以将证据表示成如下：

$$p(y, M) = \int p(y \mid \theta, M) p(\theta \mid M) \mathrm{d}\theta_m$$

也就是说，为了计算出证据$p(y \mid M)$，我们需要边缘化（通过求和或者积分）所有可能的$p(\theta \mid M)$，即根据给定模型边缘化所有 θ 的先验。

$p(y \mid M)$本身没有多少信息量，就像信息准则一样，重要的是其相对值，因此，当我们希望比较两个不同模型的时候，我们会计算其证据的比例，从而得到贝叶斯因子：

$$BF = \frac{p(y \mid M_0)}{p(y \mid M_1)}$$

当 $BF > 1$ 时，模型 0 比模型 1 对数据解释得更好。有人总结出了下面的列表来表示模型 0 与模型 1 的对比。

- 1-3：微弱
- 3-10：中等

- 10-30：强
- 30-100：很强
- >100：非常强

注意，这些准则都是一些经验性的指导，最终结果一定要放在具体场景中去解释，同时还应该给出足够多的信息方便别人检查，从而确定是否同意我们的结论。得出结论所需的证据在不同场合下是不一样的。比如说你是在做粒子物理学，或是在法庭上，又或者是决定是否要撤离一个城镇以防止数百人死亡。

6.4.1　类比信息量准则

如果对贝叶斯因子求 log，我们可以将两个边缘似然的比值转换成做差，这样比较边缘似然就与前面比较信息准则类似了。不过，衡量模型对数据的拟合程度的项以及惩罚项去哪儿了呢？前者包含在了似然的部分，而后者是对先验取平均的部分。参数越多，先验空间相比似然就越大，因而平均之后似然就会较低，而且参数越多，先验就会越分散，因而在计算证据的时候惩罚越大。这也是为什么人们说贝叶斯理论会很自然地惩罚更复杂的模型，或者称贝叶斯理论自带奥卡姆剃刀。

6.4.2　计算贝叶斯因子

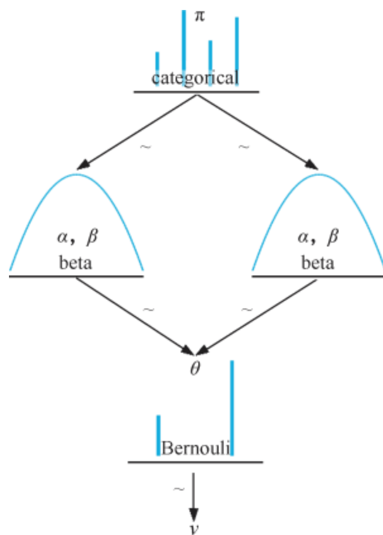
贝叶斯因子的计算可以视作分层模型的应用，其中高层的参数可以看作是从一个类别分布中采样后将序号赋给每个模型。换句话说，我们同时对两个（或多个）模型进行推断，同时用一个离散的变量在模型之间做选择。对每个模型的采样次数正比于$p(M_x \mid y)$，为了计算贝叶斯因子，我们有下式：

$$\frac{p(y \mid M_0)}{p(y \mid M_1)} = \frac{p(M_0 \mid y)p(M_1)}{p(M_1 \mid y)p(M_0)}$$

等式右边的第一项称作后验相对可能性，第 2 项称作先验相对可能性。回忆一下前面我们对相对可能性的定义。如果你好奇等式是怎么来的，根据贝叶斯理论将$p(y \mid M_0)$和$p(y \mid M_1)$展开后相除即可。为了展示贝叶斯因子的计算过程，这里再次以抛硬币问题为例。

```
coins = 30
heads =  9
y = np.repeat([0, 1], [coins-heads, heads])
```

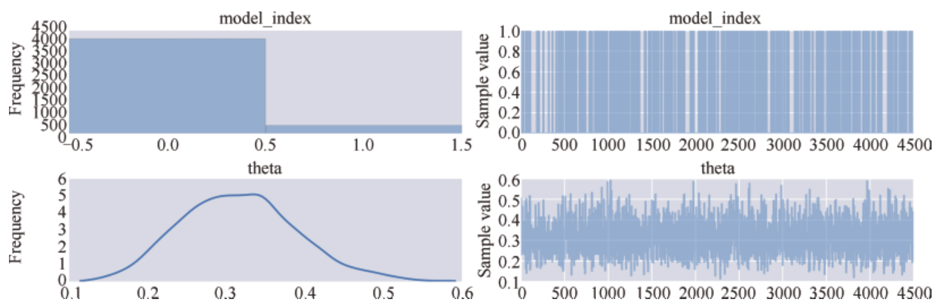
用 Kruschke 图将我们的模型表示出来，如图所示，这个例子中，我们选用了两个 beta 先验：一个趋近于 0；另一个趋近于 1。



注意这里我们计算贝叶斯因子时比较的是模型之间先验的不同，当然，模型之间的似然或者两者同时都有可能不同，本质上思想是一致的。接下来用 PyMC3 构建模型。为了切换先验，我们使用了 `pm.switch()` 函数，如果该函数的第一个参数为 0，则返回第 2 个参数，否则返回第 3 个参数。这里我们同样使用 `pm.math.eq()` 函数去检查 model_index 变量是否为 0。

```
with pm.Model() as model_BF:
    p = np.array([0.5, 0.5])
    model_index = pm.Categorical('model_index', p=p)
    m_0 = (4, 8)
    m_1 = (8, 4)
    m = pm.switch(pm.math.eq(model_index, 0), m_0, m_1)

    theta = pm.Beta('theta', m[0], m[1])
    y = pm.Bernoulli('y', theta, observed=y)
    trace_BF = pm.sample(5000)
chain_BF = trace_BF[500:]
pm.traceplot(chain_BF)
```



现在我们可以通过变量 model_index 计算贝叶斯因子，注意我们已经有了每个模型的先验：

```
pM1 = chain_BF['model_index'].mean()
pM0 = 1 - pM1
BF = (pM0/pM1)*(p[1]/p[0])
```

最终得到的贝叶斯因子的值约为 11，也就是说，我们更倾向于使用模型 0。这个结论完全合理，因为对于$\theta = 5$的情况，正面朝上出现得更少，两个模型之间的唯一区别是模型 0 更适用于$\theta < 0.5$（反面朝上多于正面朝上），而模型 1 更适用于$\theta > 0.5$（正面朝上多于反面朝上）。

下面讲一下计算贝叶斯因子的一些常见问题。

用我们定义的方式计算贝叶斯因子会有一些问题，比如当其中一个模型比另一个模型更好时，根据定义，我们会对更好的这个模型采样次数更多，这可能会导致我们对另外一个模型欠采样。另外，第 1 个问题是：即使某些参数没有用于拟合数据，也会更新。也就是说，当模型 0 被选择时，模型 1 中的参数也会更新，不过由于这部分参数并没有用于解释数据，值受限于先验。如果先验太模糊，有可能当我们选到模型 1 时，参数值距离上一次被接受的值太远了，因而该步被拒绝，从而导致采样会出现问题。

为了避免遇到这些问题，我们对模型做了两处修改来改进采样过程。

- 理想情况下，如果两个模型都访问相同次数，我们会得到一个更好的采样，因此我们对模型的先验做出调整（前一个模型中的 p 值），从而向原来访问频次较低的模型倾斜。这个过程对贝叶斯因子的计算不会有多大影响，因为我们在计算过程中包含了先验。
- 根据 Kruschke 以及其他人的建议，可以使用伪先验，其思想很简单：当没被选择的模型的参数出现自由漂移时，可以尝试手动限制它们，不

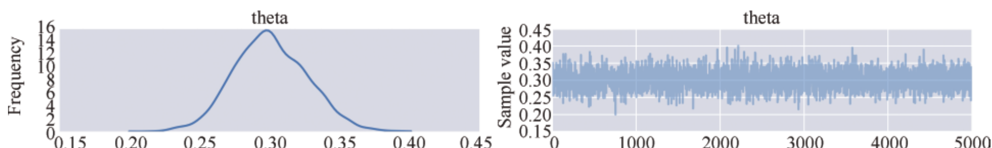
过是在这个模型没被使用的时候。你可以在 Kruschke 的书中找到使用伪先验的例子，我将对应的例子转成了 Python/PyMC3，可以查看这里：https://github.com/aloctavodia/Doing_bayesian_data_ analysis。

6.5 贝叶斯因子与信息量准则

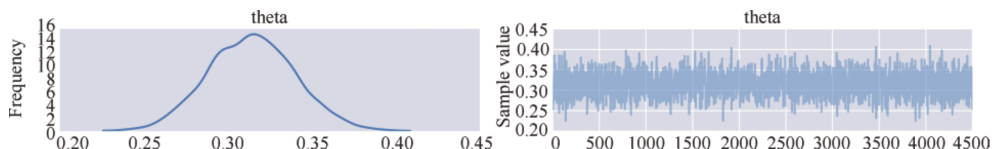
前面我们已经说过，贝叶斯因子对先验过于敏感，做推断时某些参数对推断结果几乎没有影响，但得到的贝叶斯因子却有很大区别。这也是许多贝叶斯学派的人不喜欢贝叶斯因子的原因之一。 现在我们来看一个例子，帮助我们理解什么是贝叶斯因子以及信息准则。回到抛硬币例子中定义数据的部分，现在我们将硬币数量设为 300 个，90 个正面朝上。这个比例与之前的类似，不过现在的数据量是之前的 10 倍，然后单独运行每个模型。

```
with pm.Model() as model_BF_0:
    theta = pm.Beta('theta', 4, 8)
    y = pm.Bernoulli('y', theta, observed=y)

    trace_BF_0 = pm.sample(5000)
chain_BF_0 = trace_BF_0[500:]
pm.traceplot(trace_BF_0);
```




```
with pm.Model() as model_BF_1:
    theta = pm.Beta('theta', 8, 4)
    y = pm.Bernoulli('y', theta, observed=y)
    trace_BF_1 = pm.sample(5000)
pm.traceplot(trace_BF_1);
```

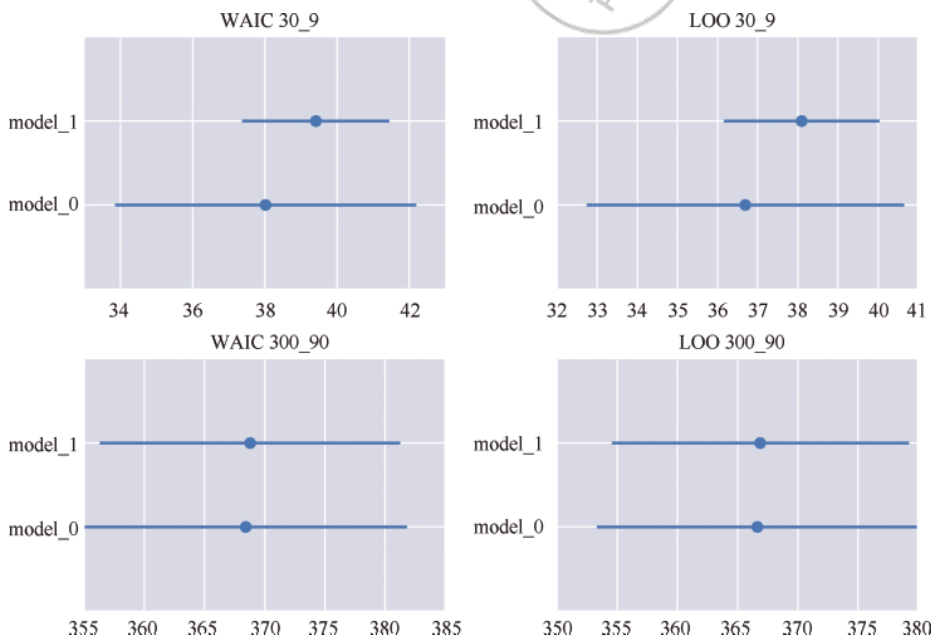


对后验进行检查，尽管二者的先验不同，可以看到两个模型的预测结果很相似。原因是我们有足够的数据，从而将先验的效果削弱了（尽管效果还在）。现在对两个模型计算贝叶斯因子，可以得到结果约为 25，这个结果意味着我们更

倾向于模型 0。可以看出，当我们增加数据个数的时候，不同模型之间的对比更明显了，这一点完全合理，因为数据更多的时候，我们更加确定模型 1 的先验假设与实际的数据不符。不过需要注意，当我们增加数据的时候，两个模型都倾向于得出相同的 θ 值，实际上两个模型得到的值都接近 0.3。因此，如果打算用 θ 做预测，那么两个模型其实没有特别大的区别。在这个例子中，贝叶斯因子告诉我们一个模型要比另外一个模型更好，某种程度上是在帮助我们找到真正的模型，不过如果只是根据两个模型估计出来的参数做预测，二者的结果差不多。

再比较下 WAIC 和 LOO（如图）；模型 0 和模型 1 的 WAIC 分别约为 368.4 和 368.6，LOO 分别约为 366.4 和 366.7。直观上看区别似乎很小，不过重要的是在 30 个硬币中应该有 9 个正面朝上的情况下，模型 0 和模型 1 的 WAIC 分别为 38.1 和 39.4，LOO 分别为 36.6 和 38.0。也就是说，相对差别随着数据量的增加而减少了，θ 的估计值越相似，根据信息准则得到的预测准确率的结果也越相似。这个例子应该能澄清贝叶斯因子与信息准则之间的区别了。

下图显示了 WAIC 和 LOO 以及它们的标准差，第 1 行对应抛硬币问题中的 30 个样本中 9 次正面朝上的情况；第 2 行对应 300 个样本中 90 个正面朝上的情况。



6.6 总结

本章开始我们先建立了这样一种直观感受：好的模型应该能简单有效地解释数据。基于该直觉，我们讨论了统计学和机器学习中的过拟合和欠拟合问题，首先分析了传统的 AIC，然后是与之类似的更符合贝叶斯的 DIC，接下来讨论了基于二者的改进版 WAIC。同时我们还简单讨论了经验性的交叉验证方法以及采用 LOO 对其结果进行近似的方法。本章还从另外一个角度简要讨论了先验与多层模型。最后，我们讨论了贝叶斯因子，如何计算贝叶斯因子，以及如何解决一些与之相关的采样问题。本章的最后还解释了使用贝叶斯因子与信息准则的不同目的。

6.7 深入阅读

- 《Statistical Rethinking》中的第 6 章。
- 《Doing Bayesian Data Analysis, Second Edition》中的第 10 章。
- 《Bayesian Data Analysis, Third Edition》中的第 7 章。
- Jake VanderPlas 关于模型选择的一篇博客：http://jakevdp.github.io/blog/2015/08/07/frequentism-and-bayesianism-5-model-selection/。
- 用 LOOCV 和 WAIC 对贝叶斯模型进行评估：http://arxiv.org/abs/1507.04544。

6.8 练习

（1）本题与正则先验有关。在生成数据部分的代码中，将 `order=2` 改成其他值，比如 `order=5`，然后拟合 model_p 并画出结果的曲线。重复该过程，用 st=100 的 beta 分布替换 st=1 的 beta 分布并画出结果曲线。两种情况下的曲线有什么不同？如果换做 sd=np.array([10, 0.1, 0.1, 0.1, 0.1]) 呢？

（2）重复上面的练习，将数据的个数增加到 500。

（3）用 3 阶的数据去拟合，计算 WAIC 和 LOO，画出结果，将它们与线性和抛物线模型进行比较。

（4）用 pm.sample_ppc() 重跑 PPC 的例子，不过画出 y 的值而非其均值。

（5）阅读并运行 PyMC3 官方文档中后验预测的例子：https://pymc-devs.github.io/pymc3/notebooks/posterior_predictive.html。特别留意 Theano 中共享变量的使用。

（6）分别用均匀先验 beta(1,1) 和其他先验（如 beta(0.5,0.5)）比较抛硬币问题中的贝叶斯因子。将数据个数设为 30，其中 15 次正面朝上。将推断结果与第 1 章中得到的结果进行比较。

（7）重复本章最后一个例子，减少样本大小，然后再比较贝叶斯因子与信息量准则。



第 7 章 混合模型

一种常见的构建模型的方式是将简单的模型组合在一起得到一个更复杂的模型。在统计学中，这类模型通常称为混合模型。混合模型有许多用途，比如直接对亚种群建模，或者作为一个小技巧来处理无法用简单分布描述的复杂概率分布。这一章我们将学习如何构建这类混合模型，同时还将看到，前面几章中的一些模型本质上其实也是混合模型，而在这一章中，我们将从混合模型的视角去揭示它们。

本章我们将学习以下内容：

- 有限混合模型；
- 零膨胀泊松分布；
- 零膨胀泊松回归；
- 鲁棒逻辑回归；
- 基于模型的聚类；
- 连续混合模型。

7.1 混合模型

有时候，某种现象或者过程无法用一个简单的分布（比如高斯分布、伯努利分布或者其他常见的概率分布）描述，但可以使用这些分布的组合来描述。这种假设数据是从一系列分布的组合中得到的模型，称作**混合模型**。

一种应用混合模型的场景是：数据集来自于一些亚种群的组合。比如，在描述身高分布的时候，我们会很自然地将成人分为男人和女人两个亚种群，此外，如果想要在分析中包含未成年人，可能需要加入第 3 个亚种群去描述未成年人（在这个种群中可能不需要区分男孩和女孩）。另外一个有关混合模型的经典例子是用其来描述一组手写数字（MNIST 数据集），这种情况下，可以很自然地使用

10 个亚种群对数据建模。

某些情况下，选择混合模型可能是出于数学上（或计算上）方便性的考虑，而不是因为我们希望尽量用亚种群去描述数据。以高斯分布为例，许多单峰分布和近似对称的分布都可以用高斯分布去近似，不过对于多峰分布或者偏斜分布呢？我们还可以使用高斯分布对其建模吗？答案是如果使用混合高斯分布的话是可以的。在高斯混合模型中，每个高斯分布都有不同的均值以及相同（或不同）的标准差。通过组合高斯分布，我们可以给模型增加灵活性，从而适应复杂的数据分布。事实上，我们可以用多个高斯分布去拟合任意分布，而不论该分布有多么复杂或者奇怪，具体的高斯分布个数取决于拟合的精度和数据的具体分布情况。充分应用该思想的一个例子是前面见过的 KDE 方法，用来画出数据的分布（而不是用直方图），该方法对每个点都应用一个分布（在 Scipy 的实现中用的是高斯分布），然后将每个单独的高斯分布都求和之后用来拟合数据的分布。KDE 属于非参的方法，我们将在第 8 章详细讨论非参方法。现在先只关注如何用混合高斯模型来拟合任意分布。

不论我们是真的相信数据中存在亚种群，还是为了利用其数学简洁性（又或者是二者皆有之），混合模型都是一种不错的选择，使用混合分布去描述数据可以给我们的模型增加很大的灵活性。

7.1.1　如何构建混合模型

构建有限混合模型的思想是：我们有一定数量的亚种群，每个亚种群都可以用一种分布来表示，然后还有符合这些分布的数据点，不过我们并不知道每个点具体属于哪个分布，因而我们希望正确地区分每个点。这类问题可以通过多层模型来实现，在模型的上层有一个随机变量，通常称为隐层变量（该变量一般无法直接观测到），然后根据隐变量的函数来确定每个观测点属于哪个分布。也就是说，隐变量决定使用哪个分布对数据点进行建模。在文献中，人们经常使用字母 z 来表示隐变量。

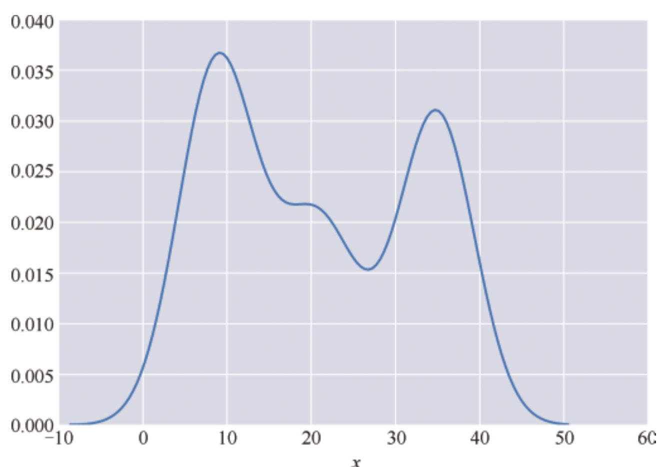
让我们从一个简单的例子来着手构建混合模型。假设我们有一个数据集，希望用 3 个高斯变量对其描述。

```
clusters = 3
n_cluster = [90, 50, 75]
```

```
    n_total = sum(n_cluster)
    means = [9, 21, 35]
    std_devs = [2, 2, 2]

    mix = np.random.normal(np.repeat(means, n_cluster), np.repeat(std_devs,
n_cluster))


    sns.kdeplot(np.array(mix))
    plt.xlabel('$x$', fontsize=14)
```



在许多真实场景中，当我们想着手构建模型的时候，通常从更简单有效的模型开始，然后再逐步变复杂。这种做法的优点是：可以一步步建立对问题和数据的直观认识，同时避免一上来就被复杂的模型难倒了（复杂的模型通常更难以调试）。

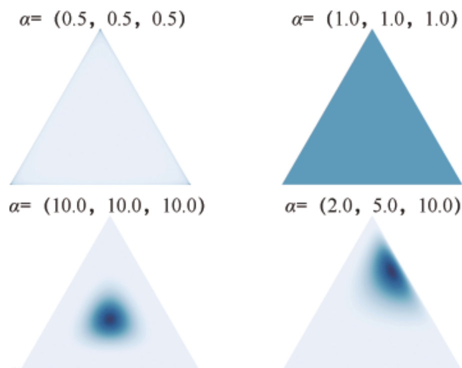
因此，先假设我们的数据可以用 3 个高斯分布来描述（或者更一般地，设为 k 个高斯分布）。做出该假设的原因可能是我们已经有足够的实验或理论知识做支撑，也可能是我们通过直接观察得出了该假设。接下来我们还假设已经知道了每个高斯分布的均值和标准差。

有了这些假设之后，问题简化成了将每个数值归类到 3 个已知的高斯分布中。有许多方法可以解决这个问题，显然，这里我们选择用贝叶斯的方法解决，并构建一个概率模型。

这里借用抛硬币问题中的思想来构建模型。在抛硬币问题中，可以使用伯努

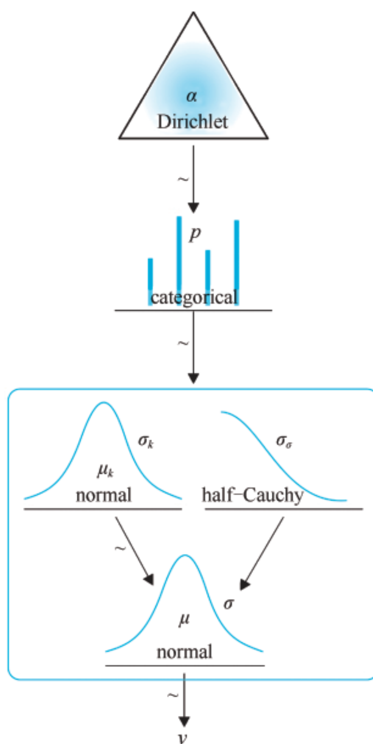
利分布来描述可能出现的结果。由于我们不知道正面朝上或者反面朝上的概率，于是采用了一个 beta 先验分布来描述。现在高斯混合模型遇到的问题也很相似，只不过现在我们有 k 个结果。

伯努利分布更一般的形式是类别分布，beta 分布的一般形式是**狄利克雷分布**。狄利克雷分布初看可能有点奇怪，因为它是一种单纯形，有点像 n 维的三角形：1- 单纯形是一个线段；2- 单纯形是一个三角形；3- 单纯形是正四面体，以此类推。为什么是一个单纯形？直观上看，这是因为该分布是一个长度为 k 的向量，其中每个元素都为正数而且所有元素之和为 1。在理解为何狄利克雷分布是 beta 分布的一般形式之前，我们先回顾下 beta 分布的一些特性。前面我们用 beta 分布来描述有两种结果的问题，其中之一的概率为 p，另一个概率为 1-p，因而我们可以认为 beta 分布返回的是一个长度为 2 的向量，[p, 1-p]。当然，在实践中，我们通常忽略了 1-p 这一项，因为它已经通过 p 完全被定义了。此外，beta 分布也可以通过两个标量 α 和 β 来定义。这些参数如何类比到狄利克雷分布中呢？我们先考虑下最简单的狄利克雷分布，用来对有 3 种结果的问题建模。此时的狄利克雷分布返回一个长度为 3 的向量 [p,q,r]，其中 $r=1-(p+q)$。我们也可以用 3 个标量来参数化地描述狄利克雷分布，可以称为 α、β 和 γ，不过这不方便推广到更高维的情况，所以这里使用一个长度为 k 的向量 α 来描述，其中 k 对应可能出现的结果的种类个数。我们可以将 beta 分布和狄利克雷分布想象成描述概率分布的分布，为了更直观地理解，我们将不同参数对应的狄利克雷分布画了出来，留意下图中的每个三角形与参数相近的 beta 分布之间的联系。



上图是在 Thomas Boggs 代码的基础上稍做修改后生成的，你可以在本书相关的代码中找到源码，在本章阅读更多部分可以了解更多细节。

现在我们对狄利克雷分布有了更多了解，也就意味着掌握了构建混合模型所需要的全部基础。一种可视化的方法是：将其看作是基于高斯估计模型的（k 面）抛硬币问题，当然，你可能更愿意将其看作（k 面）掷骰子问题。用 Kruschke 图可以将模型画成如下形式：



这里圆角矩形框表示我们有 k 个高斯似然（以及对应的先验），类别变量决定具体使用哪一个描述数据。

记住，我们假设已经知道了这些高斯分布的均值和标准差；只需要将每个点赋给一个高斯分布即可。下面构建模型中的一个细节是，我们已经有了两个采样器，Metropolis 和 ElemwiseCategorical，后者是专门为离散变量的采样设

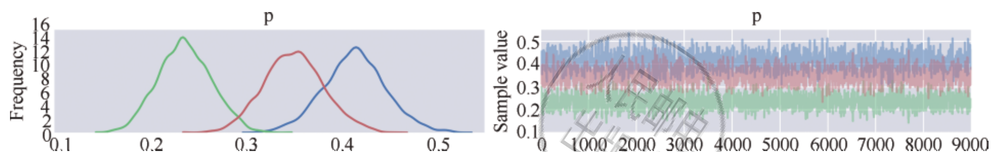
计的。

```
with pm.Model() as model_kg:
    p = pm.Dirichlet('p', a=np.ones(clusters))
    category = pm.Categorical('category', p=p, shape=n_total)

    means = pm.math.constant([10, 20, 35])

    y = pm.Normal('y', mu=means[category], sd=2, observed=mix)

    step1 = pm.ElemwiseCategorical(vars=[category], values=range(clusters))
    step2 = pm.Metropolis(vars=[p])
    trace_kg = pm.sample(10000, step=[step1, step2])
      chain_kg = trace_kg[1000:]
       varnames_kg = ['p']
       pm.traceplot(chain_kg, varnames_kg)
```



现在我们已经知道了高斯混合模型的基本结构，接下来我们再添加一层更复杂的，估计出高斯分布的参数。我们假设有 3 个不同的均值和一个共享的标准差。

和往常一样，模型可以很容易地用 PyMC3 的语法来描述。

```
with pm.Model() as model_ug:
    p = pm.Dirichlet('p', a=np.ones(clusters))
    category = pm.Categorical('category', p=p, shape=n_total)

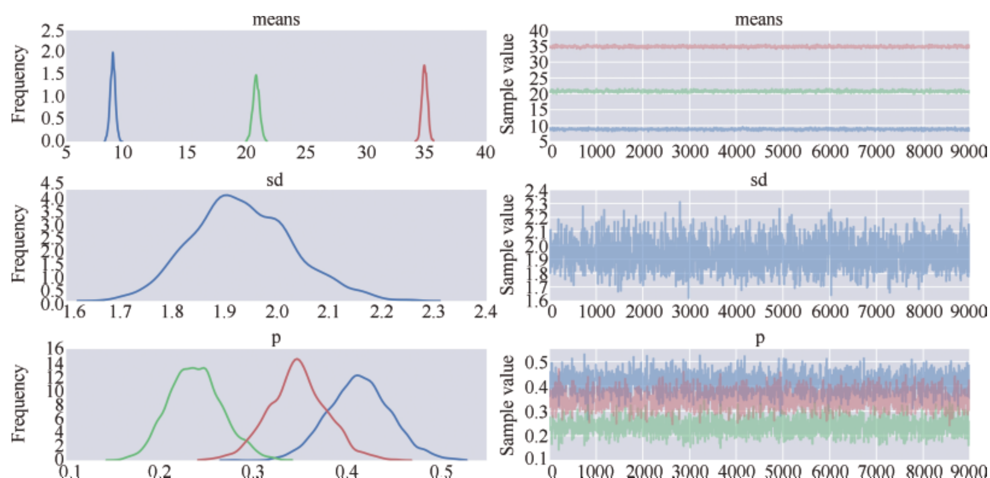
    means = pm.Normal('means', mu=[10, 20, 35], sd=2, shape=clusters)
    sd = pm.HalfCauchy('sd', 5)

    y = pm.Normal('y', mu=means[category], sd=sd, observed=mix)

    step1 = pm.ElemwiseCategorical(vars=[category], values=range(clusters))
    step2 = pm.Metropolis(vars=[means, sd, p])
    trace_ug = pm.sample(10000, step=[step1, step2])
```

然后再看一下迹。

```
chain = trace[1000:]
varnames = ['means', 'sd', 'p']
pm.traceplot(chain, varnames)
```



以及推断总结的表格。

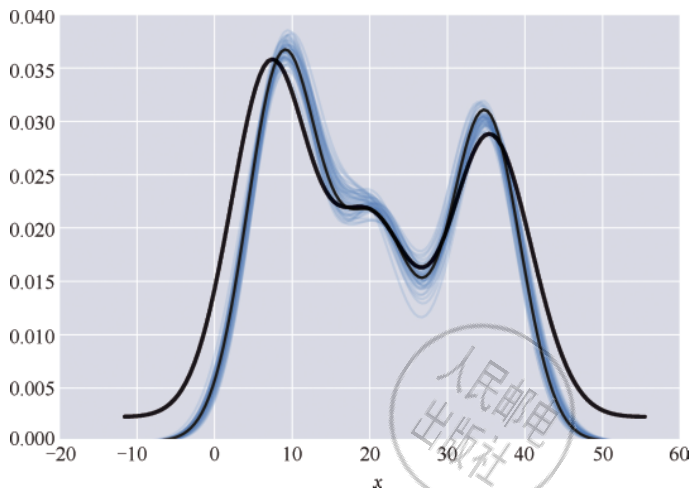
```
pm.df_summary(chain, varnames)
```

	mean	sd	mc_error	hpd_2.5	hpd_97.5
means__0	21.053935	0.310447	0.012280	20.495889	21.735211
means__1	35.291631	0.246817	0.008159	34.831048	35.781825
means__2	8.956950	0.235121	0.005993	8.516094	9.429345
sd	2.156459	0.107277	0.002710	1.948067	2.368482
p__0	0.235553	0.030201	0.000793	0.179247	0.297747
p__1	0.349896	0.033905	0.000957	0.281977	0.412592
p__2	0.347436	0.032414	0.000942	0.286669	0.410189

接下来再做后验预测检查，看看模型从数据中学到了什么。

```
ppc = pm.sample_ppc(chain, 50, model)
for i in ppc['y']:
    sns.kdeplot(i, alpha=0.1, color='b')

sns.kdeplot(np.array(mix), lw=2, color='k')
plt.xlabel('$x$', fontsize=14)
```



注意查看图中颜色较浅的蓝色线条所表示的不确定性，从中可以看到，在 x 较小或者较大的值附近，不确定性较低，而在中间部分不确定性较高。这一点符合直观认识，因为不确定性较高的区域对应高斯分布重叠的地方，因而比较难以分辨一个点具体属于哪一个高斯分布。我承认，这个问题比较简单而且不那么有挑战性，不过该问题有助于强化我们的直觉，即一个简单的模型可以扩展并应用到复杂问题中。

7.1.2　边缘高斯混合模型

前面的模型中，我们显式地定义了隐变量 z，这样做带来的一个问题是，对离散的隐变量 z 采样通常会导致模型收敛较慢而且不利于探索分布的尾部。使用一些针对离散隐变量特殊定制的采样器能有助于改进采样过程，除此之外，还有一种做法是换一种参数化表示方法，重新构建一个等价的模型。注意，在混合模

型中，观测变量 z 是根据隐变量 z 构建的，即：

$$p(y\,|\,z,\theta)$$

我们可以认为隐变量 z 是一个可以被边缘化的多余变量，因而得到：

$$p(y\,|\,\theta)$$

幸运地是，PyMC3 包含一个分布可以高效地处理该过程，不需要我们手动处理中间的数学过程，因此我们可以按如下形式构造高斯混合模型。

```
with pm.Model() as model_mg:
    p = pm.Dirichlet('p', a=np.ones(clusters))

    means = pm.Normal('means', mu=[10, 20, 35], sd=2, shape=clusters)
    sd = pm.HalfCauchy('sd', 5, shape=clusters)

    y = pm.NormalMixture('y', w=p, mu=means, sd=sd, observed=mix)

    step = pm.Metropolis()
    trace_mg = pm.sample(2000, step)
```

这里将运行模型、画图以及探索结果的过程留给读者当做练习。

7.1.3　混合模型与计数类型变量

有时候，我们会用到一些计数类型的数据，比如放射性核的衰减、每对夫妻的小孩个数或者是推特的粉丝个数。这些例子的相同之处是，我们可以用非负的离散数字来建模 {0, 1, 2, 3...}。这类变量称作计数变量，通常使用泊松分布对其建模。

泊松分布

假设希望统计一条马路上每个小时经过的红色小汽车数量，我们就可以选用泊松分布来描述该数据。泊松分布通常用来描述在一段固定的时间（区间）内某一特定事件发生的次数。因而泊松分布假设在这段固定的时间（区间）内，事件之间的发生是相互独立的。该离散分布只有一个参数 λ，对应分布的均值和方差。泊松分布的概率质量函数为：

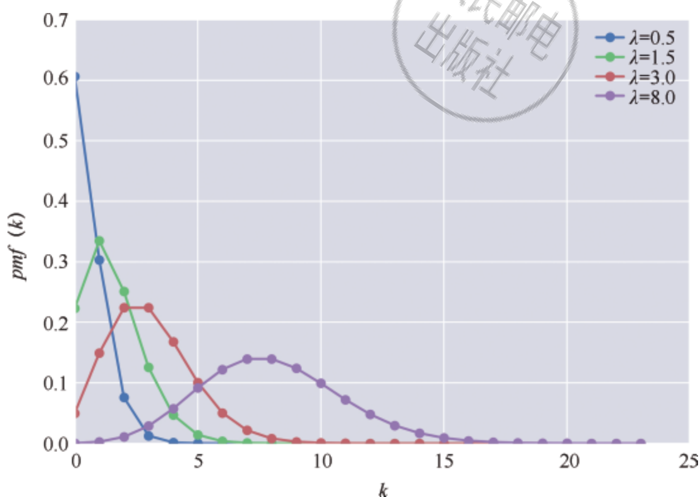
$$pmf(k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

其中：

- λ 是单位时间（空间）内时间发生的均值
- k 是正整数，如 0，1，2，…
- $k!$ 表示 k 的阶乘

下面的图中，我们可以看到一系列不同 λ 值对应的泊松分布的例子。

```
lam_params = [0.5, 1.5, 3, 8]
k = np.arange(0, max(lam_params) * 3)
for lam in lam_params:
    y = stats.poisson(lam).pmf(k)
    plt.plot(k, y, 'o-', label="$\\lambda$ = {:3.1f}".format(lam))
plt.legend()
plt.xlabel('$k$', fontsize=14)
plt.ylabel('$pmf(k)$', fontsize=14)
```



注意，λ可以是一个浮点数，而 k 只能是正整数。前面的图中，每个点表示对应 x 取值的分布情况，这里点与点之间用曲线连接起来是为了帮助我们理解该分布的形状，记住，泊松分布是离散的分布。泊松分布可以看作是二项分布的一种特殊形式（当 n 非常大，而 p 很小时）。这里不深入过多的数学细节，我们可

以这么思考，由于可能看到或者看不到红色小汽车，我们可以用一个二项分布来对红色小汽车的个数建模，此时我们有：

$$x \sim Bin(n, p)$$

然后，二项分布的均值为：

$$E[x] = np$$

二项分布的方差为：

$$Var[x] = np(1-p)$$

不过考虑到即使你站在一条非常繁忙的马路上，看到红色小汽车的机会相比于整个城市的汽车总量仍然很少，因而我们有：

$$n \gg p \Rightarrow np \simeq np(1-p)$$

于是，我们可以做如下近似：

$$Var[x] = np$$

现在均值和方差是用同一个数来表示的，于是我们可以有信心地说这里的变量服从泊松分布，此时我们有：

$$x \sim Pois(\lambda = np)$$

零膨胀泊松模型

当我们计数时，数字 0 可能会多次出现，原因有多个，有可能在我们统计红色汽车个数时刚好没有车通过那条马路，或者是我们碰巧错过了（有可能我们没有注意到大卡车后面的红色小汽车）。因此，如果使用的是泊松分布，在进行后验预测检查时，不会得到一个非常漂亮的拟合，因为如果数据真的符合泊松分布的话，我们不会看到这么多的 0。

如何解决这个问题呢？我们可以尝试找到模型预测比观察到的 0 更少的原因，并将其加入到模型中。不过大多数情况下，只需要假设我们有一个由泊松分布组成的混合模型即可，其中泊松分布的概率为ψ，取到额外 0 的概率为$1-\psi$。如果选用混合模型，那么我们得到的就是零膨胀泊松模型（Zero-Inflated Poisson，

ZIP）。在有些书中，你也许会看到ψ表示额外的 0，而$1-\psi$表示泊松分布的概率。这个问题不太大，只需要在具体例子中注意区分就好。一个基本的零膨胀泊松模型的表示形式如下：

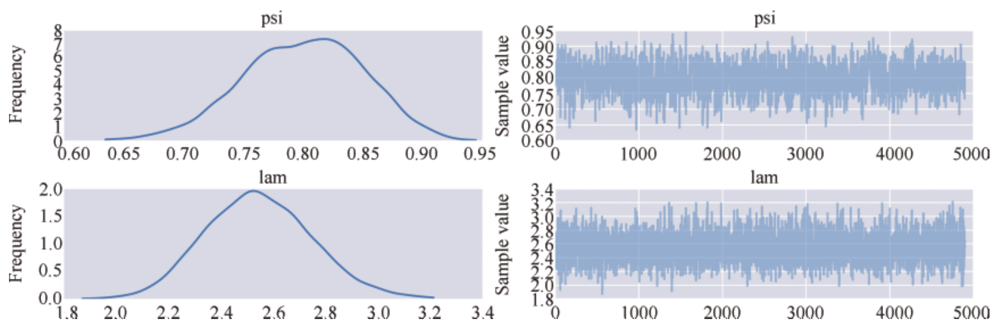
$$p(y_j = 0) = 1 - \psi + (\psi)\mathrm{e}^{-\lambda}$$

$$p(y_j = k_i) = \psi \frac{\lambda^{k_i} \mathrm{e}^{-\lambda}}{k_i!}$$

其中$1-\psi$是取到额外 0 的概率。 我们可以根据这些等式用 PyMC3 来构建模型，不过 PyMC3 已经自带了 ZIP 分布，因此这个模型写起来更容易一些。由于 Python 已经有一个匿名函数关键字 lambda 了，因此我们这里使用了 lam 来表示上式中的λ，于是我们可以用以下 PyMC3 代码实现 ZIP 模型：

```
with pm.Model() as ZIP:
    psi = pm.Beta('p', 1, 1)
    lam = pm.Gamma('lam', 2, 0.1)

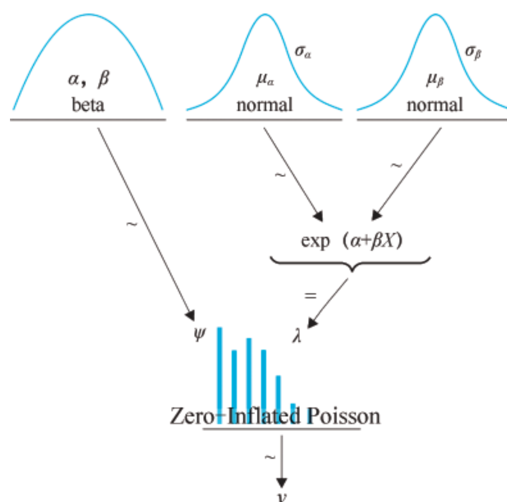
    y = pm.ZeroInflatedPoisson('y', lam, psi, observed=counts)
    trace = pm.sample(1000)
pm.traceplot(trace[:]);
```

泊松回归与ZIP回归

ZIP 模型看起来可能有点笨，不过有时候我们需要用它来估计一些简单的分布（甚至是泊松分布或者高斯分布）。此外，泊松分布或者 ZIP 分布还可以用于线性模型。在第 4 章中，我们看到过，一元线性回归模型可以由一个线

性模型（通过一个恒等逆连结函数）和作为噪声或者误差的高斯分布（或 t 分布）组成。第 5 章中，我们还看到了如何将这个模型用于分类，其中使用了逻辑函数或者 softmax 函数作为逆连结函数，同时用伯努利分布或者类别分布对因变量建模。根据同样的思想，当因变量为计数变量时，可以用一个泊松分布或者 ZIP 分布进行回归分析，下图就是一个 ZIP 回归，不过如果我们要做泊松回归就不需要包含ψ，因为我们不需要对多余零建模。注意，现在我们可以使用指数函数作为逆连结函数，该选择保证了线性模型的返回值始终都是正的。



为了用例子说明 ZIP 回归模型是如何实现的，这里我们使用从 http://www.ats.ucla.edu/stat/data/fish.csv 得到的数据集，该数据集也可以在本书的代码中找到。 问题描述如下：我们在一个公园管理处工作，希望提升游客的体验，因此我们决定对访问公园的 250 组游客进行问卷调查，在收集到的数据中包含以下内容：

- 他们抓到的鱼的条数；
- 每组中有多少个小孩；
- 他们是否带了露营车到公园。

我们将使用该数据，根据小孩个数和是否带露营车这两个变量来预测它们抓

到的鱼的条数。下面用 Pandas 通过如下代码载入数据：

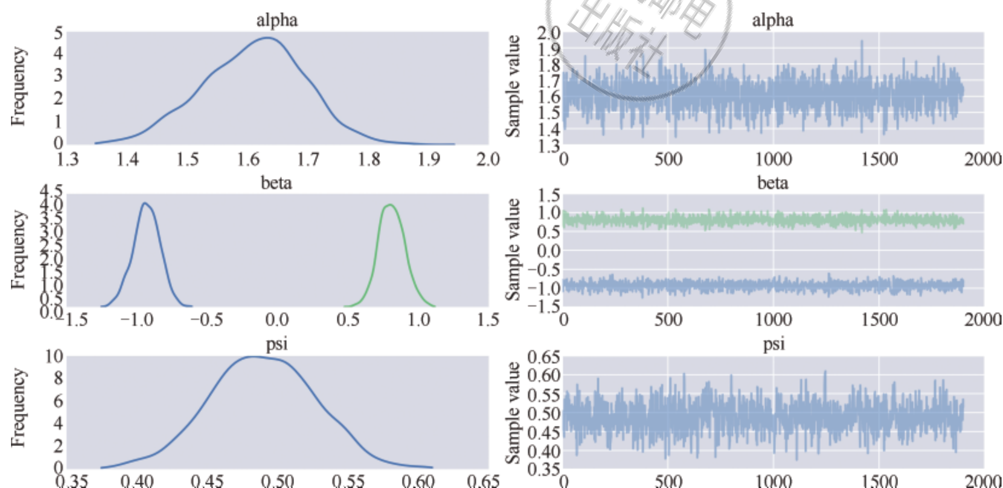
```
fish_data = pd.read_csv('fish.csv')
```

这里将探索数据集的过程当做练习留给读者，你可以自由使用画图函数或者 Pandas 中的 describe() 函数等。我们可以按如下形式先将 Kruschke 图转换成 PyMC3 中的代码：

```
with pm.Model() as ZIP_reg:
    psi = pm.Beta('psi', 1, 1)

    alpha = pm.Normal('alpha', 0, 10)
    beta = pm.Normal('beta', 0, 10, shape=2)
    lam = pm.math.exp(alpha + beta[0] * fish_data['child'] + beta[1] *
fish_data['camper'])

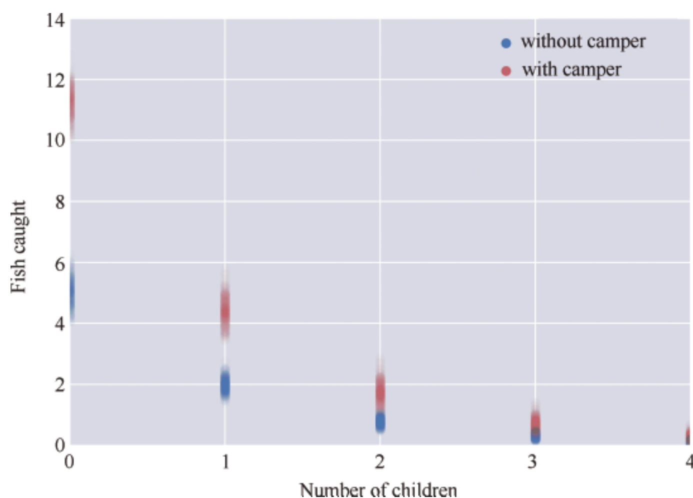
    y = pm.ZeroInflatedPoisson('y', lam, psi, observed=fish_data['count'])
    trace_ZIP_reg = pm.sample(1000)
chain_ZIP_reg = trace_ZIP_reg[100:]
pm.traceplot(chain_ZIP_reg)
```

和往常一样，你需要对采样过程做完整的检查，为了更好地理解推断结果，可以用如下代码画一幅图出来：

```
children =  [0, 1, 2, 3, 4]
fish_count_pred_0 = []
```

```
fish_count_pred_1 = []
thin = 5
for n in children:
    without_camper = chain_ZIP_reg['alpha'][::thin] + chain_ZIP_reg['beta']
[:,0][::thin] * n
    with_camper = without_camper + chain_ZIP_reg['beta'][:,1][::thin]
    fish_count_pred_0.append(np.exp(without_camper))
    fish_count_pred_1.append(np.exp(with_camper))
```

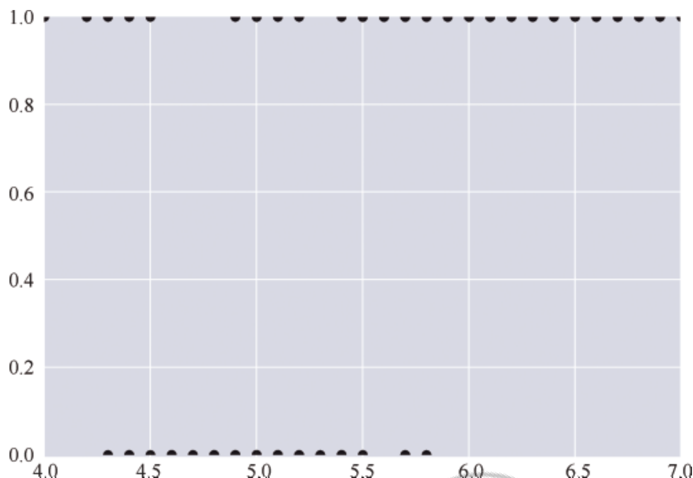


7.1.4 鲁棒逻辑回归

前面我们已经知道了如何在不直接对多余零建模的情况下，解决零过剩的问题。根据 Kruschke 的建议，另外一种类似的方法是，构建一个更鲁棒的逻辑回归。在逻辑回归中，我们将数据建模成二项式的（即 0 和 1），因而数据集有可能包含多余的 0 或者 1。这里还是以前面的鸢尾花数据集为例，不过增加了一些异常点，代码实现如下：

```
iris = sns.load_dataset("iris")
df = iris.query("species == ('setosa', 'versicolor')")
y_0 = pd.Categorical(df['species']).codes
x_n = 'sepal_length'
x_0 = df[x_n].values
y_0 = np.concatenate((y_0, np.ones(6)))
```

```
x_0 = np.concatenate((x_0, [4.2, 4.5, 4.0, 4.3, 4.2, 4.4]))
x_0_m = x_0 - x_0.mean()
plt.plot(x_0, y_0, 'o', color='k')
```



其中有一些 versicolors 的花萼长度异常短，这里可以通过一个混合模型来修复。我们假设变量的输出源自两个部分，其一是概率为 π 的随机结果，另外一个是概率为 $1-\pi$ 的逻辑回归分布。对应的数学形式如下：

$$p = 0.5\pi + (1-\pi)logistic(\alpha + \beta X)$$

注意当 $\pi = 1$ 时，$p = 0.5$；当 $\pi = 2$ 时，就得到了逻辑回归。可以在第 5 章模型实现的基础上实现上面的模型，代码实现如下：

```
with pm.Model() as model_rlg:
    alpha_tmp = pm.Normal('alpha_tmp', mu=0, sd=100)
    beta = pm.Normal('beta', mu=0, sd=10)

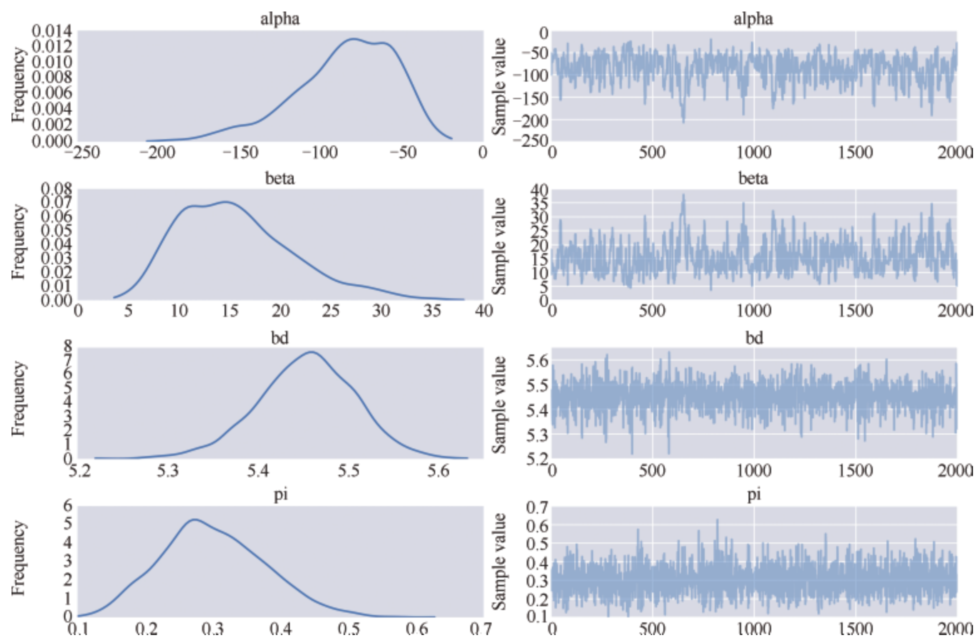
    mu = alpha_tmp + beta * x_0_m
    theta = pm.Deterministic('theta', 1 / (1 + pm.math.exp(-mu)))

    pi = pm.Beta('pi', 1, 1)
    p = pi * 0.5 + (1 - pi) * theta

    alpha = pm.Deterministic('alpha', alpha_tmp - beta * x_0.mean())
    bd = pm.Deterministic('bd', -alpha/beta)
```

```
    yl = pm.Bernoulli('yl', p=p, observed=y_0)

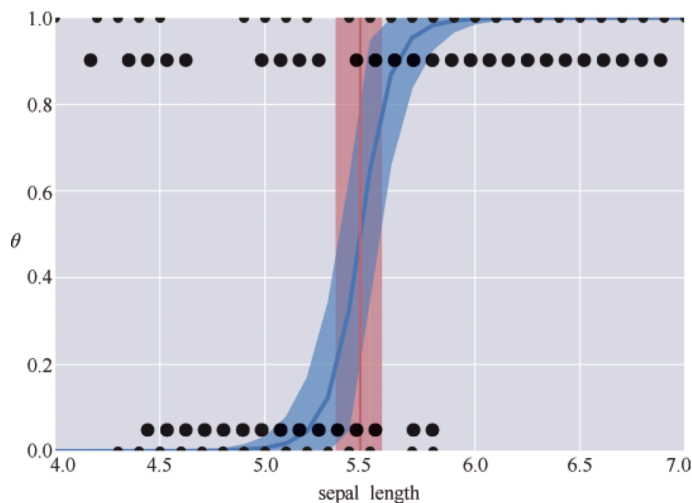
    trace_rlg = pm.sample(2000, start=pm.find_MAP())
varnames = ['alpha', 'beta', 'bd', 'pi']
pm.traceplot(trace_rlg, varnames)
```



如果将以上结果与第 5 章中的结果进行比较，我们可以看到二者差不多：

```
pm.df_summary(trace_rlg, varnames)
```

	mean	sd	mc_error	hpd_2.5	hpd_97.5
alpha	-88.04	32.06	1.99	-151.35	-32.68
beta	16.15	5.87	0.36	6.21	27.95
bd	5.45	0.05	0.00	5.34	5.55
pi	0.30	0.07	0.00	0.15	0.45



7.2　基于模型的聚类

聚类是统计学和机器学习中无监督学习的一部分，与分类有点类似，不过更复杂一些，因为聚类问题中我们没有正确的标签。

笼统地说，聚类就是在没有标签的情况下将属性相近的数据点归类到一起，使得组内的数据距离较近，而组间数据的距离较远。聚类有许多应用，比如，在种系遗传学（生物学中研究不同物种之间的进化关系的一门学科）中，聚类可以用来指导解决一些进化问题；另外一个更偏商业的应用是，根据一个用户的历史消费记录，将其与其他用户聚类到一组，从而猜出他们可能对哪些电影、书籍、歌曲等感兴趣。在一些其他无监督学习的任务中，我们可能希望直接做聚类，或者是将其作为探索式数据分析的一部分。接下来将学习一些常见的聚类准则，用于判断两个点是否属于同一个组。

通常我们使用欧几里得距离来衡量两个点之间的距离。直观上看，欧几里得距离就是连接两个点的线段长度，尽管我们测量的不是实际的物理距离（比如说米或者光年），仍然可以定义出欧几里得距离，对于 n 维空间中的两个点 q_i 和 p_i，可以将欧几里得距离定义成如下形式：

$$d(p,q)=d(q,p)=\sqrt{\sum(q_i-p_i)^2}$$

在机器学习中有一个算法叫做 K- 均值聚类，使用的就是欧几里得距离。我们暂时不会深入学习这个算法，不过我建议你先自己了解下，因为这个算法是聚类问题中最经典的一个例子，而且很容易理解、解释并实现。在 Sebastian Raschka 写的《Python Machine Learning》一书中你可以了解更多有关该算法的内容。

计算欧几里得距离或其他类似的距离并非解决聚类问题的唯一方法。另外一种方法是从概率的角度来看待聚类问题，假设数据是从某些概率分布中抽样得到的，然后可以得到一个概率模型，这类方法通常称为基于模型的聚类。

在使用贝叶斯的框架解决聚类问题的时候，混合模型是一个很自然的选择。事实上，本章第一个例子就是使用的混合模型对（未标注的）亚种群进行建模。利用概率模型可以计算出每个点属于各个类别的概率，这称作**软聚类**，与之对应的是**硬聚类**（每个点属于某个类别的概率要么是 0，要么是 1）。当然，我们可以通过某种规则或者阈值将软聚类转换成硬聚类，就像逻辑回归一样，一种做法是，计算每个点属于各个类别的概率，然后将概率最大的那个类别标签赋给该点。

7.2.1 固定成分聚类

本章的第一个例子就是将混合模型应用到分类问题的一个典型例子，我们对数据有一个经验性的假设分布，然后用 3 个高斯分布来描述数据中的亚种群。注意这里假设数据服从混合高斯分布，而不是从原来数据之间相似性测度的角度来思考的。

7.2.2 非固定成分聚类

在一些问题，比如手写数字的识别中，我们很容易就能看出类别的个数；对于另外一些问题，我们可以有一些靠谱的猜测，比如对于某个与鸢尾花数据集相似的数据集，我们可能知道其中样本的来源是一个只有 3 种鸢尾花生长的地方；而对于一些其他问题而言，预先指定类别个数的先验可能会有问题。一种贝叶斯的解决办法是使用一种非参的方法来估计出类别的个数，比如我们可以使用狄利克雷过程。在第 8 章中，我们将学习非参统计学；不过，本书并没有包含

狄利克雷过程，建议你在阅读完第 8 章之后，阅读并运行以下代码 https://pymc-devs.github.io/pymc3/notebooks/dp_mix.html，这个介绍是由 Austin Rochford 写的（PyMC3 的开发者之一，也是本书的英文版的审稿人）。

7.3　连续混合模型

本章重点关注了离散的混合模型，不过我们也可以有连续的混合模型。事实上我们已经了解了其中的一部分，连续混合模型之一就是前面介绍过的鲁棒逻辑回归模型，由两个部分组成：一个逻辑回归和一个随机估计。注意此时参数 π 不再是一个开关，而更像是一个球形把手控制着逻辑回归和随机变量的比例，只有当 π 为 0 或 1 时，我们才会得到完全随机的结果或者完全的逻辑回归结果。

多层模型也可以看做是连续混合模型，其中每个组的参数来自于上层的连续分布。更具体点，假设我们要对几个不同的组做线性回归，可以假设每个组都有自己的斜率或者所有组都共享相同的斜率。此外，除了将问题看作这两个极限情况，我们还可以构建一个多层模型，对这些极限值构建一个连续混合模型，此时这些极限值不过是分层模型中的一些特例罢了。

7.3.1　beta- 二项分布与负二项分布

beta- 二项分布是一个离散分布，通常用来描述 n 次伯努利实验中成功的次数 y，其中每次实验成功的概率 p 未知，并且假设其服从参数为 α 和 β 的伯努利 beta 分布，对应的数学形式如下：

$$BetaBinonial(y \mid n, \alpha, \beta) = \int_0^1 Bin(y \mid p, n) Beta(p \mid \alpha, \beta) \mathrm{d}p$$

也就是说，为了找到观测到结果 y 的概率，我们遍历所有可能的（连续的）p 值然后求平均。因此，beta- 二项分布也可以看作是连续混合模型。

如果你觉得 beta- 二项分布听起来很熟悉，那一定是因为你对本书的前两章学得很认真！在抛硬币的问题中，我们用到了该模型，尽管我们当时显式地使用了一个 beta 分布和一个二项分布，你也可以直接使用混合 beta- 二项分布。

类似的，还有负二项分布，可以将其看作是一个连续混合的**伽马泊松分布**，也就是将一个来自伽马分布的值作为泊松分布的参数，然后对泊松分布连同伽马分布求均值（积分）。该分布常用来解决计数型数据中的一个常见问题**过度离散**。假设你用一个泊松分布来对计数型数据建模，然后你意识到数据中的方差超出了模型的方差（使用欧松分布的一个问题是，其均值与方差是有联系的，事实上是用同一个参数描述的），那么解决该问题的一个办法是将数据看作是（连续的）泊松分布的混合，其中的泊松分布的参数来自于一个伽马分布，从而很自然地用到了负二项分布。使用混合分布之后，我们的模型有了更好的灵活性，并且能够更好地适应从数据中观测到的均值和方差。

beta- 二项分布和负二项分布都可以用作线性模型的一部分，而且也都有零膨胀的版本，此外二者都已经在 PyMC3 中实现了。

7.3.2 t 分布

前面我们介绍了 t 分布是一种更鲁棒的高斯分布。从下面的数学表达式可以看到，t 分布同样可以被看作是连续混合模型：

$$t_v(y \mid \mu, \sigma) = \int_0^\infty N(y \mid \mu, \sigma) Inv\chi^2(\sigma \mid v) \mathrm{d}v$$

注意，这个表达式与前面的负二项分布的表达式很像，不过这里是参数为 μ和 σ 的正态分布以及从参数为 v 的 $Inv\chi^2$ 分布中采样得到的 σ，也就是自由度，通常我们更倾向于称为**正态参数**。这里参数μ和 beta- 二项分布里的参数 p 概念上相似，等价于有限混合模型中的隐变量 z。对于有限混合模型来说，很多时候我们可以在推断之前先对隐变量 z 做边缘化处理，从而得到一个更简单的模型，正如前面的边缘混合模型中的例子一样。

7.4 总结

这一章中，我们学习了混合模型。混合模型可以用来解决许多问题，根据前面几章中学到的内容，我们可以很容易地构建有限混合模型。这类模型可以很方便地应用在计数类型数据的零过剩问题中，或者是在观测到过度分散的数据时用

来扩展泊松模型。另外一个应用是扩展逻辑回归，用来处理异常值。此外我们还简单地讨论了从贝叶斯角度做聚类的一些核心概念。最后本章讨论了一些有关连续混合模型的理论，以及如何将这类模型与前面几章中学到的概念联系起来，比如多层模型、t 分布。

7.5　深入阅读

- 《Statistical Rethinking》中的第 11 章。
- 《Doing Bayesian Data Analysis, Second Edition》中的第 21 章。
- 《Bayesian Data Analysis, Third Edition》中的第 22 章。

7.6　练习

1．修改本章第一个例子中的合成数据，从而使得模型更难复现出真实的参数。尝试通过修改 3 个高斯分布的均值和方差增加它们的重叠度；尝试修改每类数据的个数，并思考如何有针对性地优化模型。

2．使用“鱼”数据集扩展本章中使用的模型，将人数这一变量作为线性模型的一部分。在对多余零建模的时候包含进该变量，你应该会得到有两个线性回归的模型，其中之一将小孩个数和是否带帐篷之间的关系与泊松分布的系数联系在一起，另外一个将人数与 ψ 联系在一起。对于后者，你可能需要一个逻辑回归作为逆连结函数。

3．使用鲁棒逻辑回归中的例子，将其输入到非鲁棒逻辑回归模型中，检查异常值是否对结果有影响。你可能需要增加或者减少异常值的个数，从而更好地理解一般的逻辑回归与鲁棒逻辑回归之间的区别。

4．阅读并运行 PyMC3 中有关混合模型的例子（https://pymc-devs.github.io/pymc3/examples.html#mixture-models）。

5．用一个混合模型对三类鸢尾花进行聚类，只使用两个特征，同时假设你不知道正确的标签。你可能需要定义一个三元高斯分布，你可以从一个简单的共享的协方差矩阵开始。

第 8 章 高斯过程

目前为止我们见过的所有模型都是参数化的模型，这些模型包含固定个数的参数。此外还有一类模型称为**非参模型**，在非参模型中，参数的个数是随着数据大小变化的，换句话说，模型中潜在着无限多个参数，我们通过某种方式将其减少到某个值，从而正好能够用来描述全部数据。这一章中，我们将学习核函数的概念，以及如何从核函数的角度重新思考问题。高斯分布是统计学中的核心内容，不仅仅是对传统的方法，对于贝叶斯统计学和机器学习也是如此。我们将以高斯分布为例子，探讨如何将高斯模型的概念扩展到无限多维，并学习这类描述函数的概率分布。尽管这么做一开始看起来有点奇怪，不过好处是可以通过参数化的核函数来推断函数。

本章我们将学习：

- 非参统计；
- 核函数；
- 核函数回归；
- 高斯过程以及函数的先验。

8.1 非参统计

非参统计通常用来描述一类不依赖于参数化概率分布的统计工具 / 模型。根据这个定义，贝叶斯统计似乎不可能是非参的，因为前面我们学到过，贝叶斯统计的第一步就是在概率模型中准确地将概率分布组合在一起。第 1 章中说过，概率分布是构建概率模型的基石。在贝叶斯框架中，非参模型是指包含有无限多参数的模型，因此，我们将参数可以随着数据大小而变化的模型称作非参数化模型。对于非参数化模型而言，理论上其参数个数是无限的，实际使用中会根据数据将其收缩到一个有限的值，从而让数据本身来决定参

数的个数。

8.2　基于核函数的模型

核函数方法相关的研究是一个非常高产而且活跃的领域，有许多书讨论这个主题，其流行也是因为核函数具有一些有趣的数学特点。这里只需要知道，核函数可以作为非线性模型的基础，并且相对来说比较容易计算。比较流行的核函数方法有两种：支持向量机（Support Vector Machine，SVM）和高斯过程。后者是一种概率方法，也正是本章将要介绍的主题，前者不是概率的方法，这里不做深入讨论，你可以阅读 Jake Vanderplas 写的《Python Data Science Handbook》和 Sebastian Raschka 写的《Python Machine Learning Bayesian Analysis with Python》这两本书了解更多。在继续深入之前，我们先了解一下什么是核以及如何使用。

你可能会发现，在统计学中，核函数的定义不止一种，并且根据定义的不同，核函数的数学特性也有所不同。结合本章讨论的目的，这里将核函数看作是一个对称函数，接受两个输入并返回一个永远为正数的值，从而可以将核函数的输出看作是两个输入变量之间的相似度。

核函数有许多种，其中某些经过特殊改造后能很好地用于解决图像识别、文档分析等问题，还有些核函数适用于对周期性的函数建模。有一个核函数在统计学和机器学习中非常流行，叫做高斯核，也称作高斯径向基函数。

8.2.1　高斯核函数

高斯核函数的定义如下：

$$K(x,x') = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{w}\right)$$

其中$\|\boldsymbol{x}-\boldsymbol{x}'\|^2$称为**欧氏距离的平方**（Squared Euclidean Distance，SED），对于一个 n 维的空间，我们有：

$$\|\boldsymbol{x}-\boldsymbol{x}'\|^2=(x_1-x_1')^2+(x_2-x_2')^2+\cdots+(x_n-x_n')^2$$

注意，如果计算欧几里得距离的话需要求平方根。SED 并不满足三角不等式，因而并不是真实的距离，不过在许多常见的问题中，我们只需要对不同的 SED 进行比较，例如，计算一些点中的最小欧氏距离等价于找到最小的 SED。

有一点不太明显，高斯核函数与高斯分布的数学形式其实有一些相似之处，将常数项去掉之后，可以得到$w=2\sigma^2$，其中，w 控制着核函数的宽度，在这里正比于方差，因而有时候也称作**带宽**。

8.2.2 核线性回归

前面我们学习了线性回归模型的基本形式：

$$y=f(\boldsymbol{x})+\in$$

其中，$\in$是噪声或者误差，通常是高斯分布。

$$f(\boldsymbol{x})=\boldsymbol{\mu}=\sum_i^n\boldsymbol{\gamma}\boldsymbol{x}$$

这里我们用 $f(.)$ 来表示线性函数（没有噪声），如果使用了其他逆连结函数（比如第 5 章中的逻辑函数），我们就将它包含在 $f(.)$ 内。这里向量 $\boldsymbol{\gamma}$ 表示一个系数向量，通常是一个系数和一个或多个斜率。

在第 4 章中，我们介绍了多项式回归的概念，同时提到了，多项式回归的唯一实际用途可能就是作为统计学的教学科研工具（至少对于 2 次或 3 次以上的多项式回归）。此外我们还学习了如何将多项式回归用于非线性数据的建模过程中。

这里我们将多项式回归表示成如下形式：

$$\mu=\sum\gamma_i\phi_i(x)$$

其中，ϕ 函数表示一系列阶数逐渐增加的多项式，将向量 $\boldsymbol{x}$ 转成矩阵后，矩阵的每一列都是向量 $\boldsymbol{x}$ 的幂次（逐渐增加），从而有效地将数据映射到了更高维度的空间中，然后再从高维空间中找到一条直线去拟合数据，最后将该直线投影

回原始的低维空间时，不一定仍然是直线，而有可能是一条曲线。整个过程通常称作将输入投影到特征空间。

函数 ϕ 不一定是多项式，还有其他形式的函数将输入向量映射到（高维的）特征空间中，在此条件下，除了使用 ϕ 函数之外，我们还可以将其替换成一个核函数。尽管在数学形式上是等价的，使用核函数会让计算更容易一些，这通常称作核函数技巧。这也是为什么核函数是许多统计学方法和机器学习方法中的核心概念的主要原因，而特征空间的概念在实际中反而不那么重要了（尽管在学习核函数方法的时候还是能提供一些直观的解释）。

继续我们的讨论，这里不过多深入其中的数学细节，我们将 ϕ 替换成一个核函数 K，这里用 K 来表示高斯核函数，于是得到：

$$\boldsymbol{\mu} = \sum_i^N \gamma_i K_i(\boldsymbol{x}, \boldsymbol{x}')$$

注意，这里用 $\boldsymbol{x}$ 来表示数据，此外还有一个向量叫做 $\boldsymbol{x}'$，后者也是一个向量，通常称作**结**或者**中心点**，它均匀地分布在数据的范围内。一个特例是 $\boldsymbol{x}=\boldsymbol{x}'$，换句话说，我们完全可以将数据点作为结。

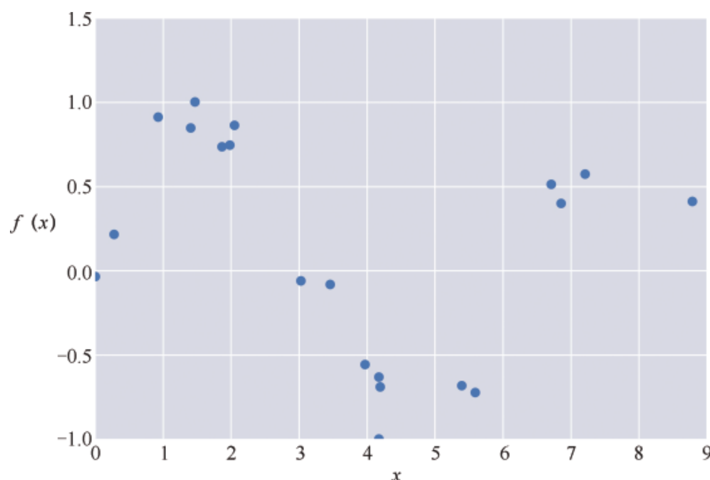
我们为什么要增加这些点呢？在直接回答这个问题之前，我们先看一下如何使用结。注意，如果结距离原数据越近，核函数的返回值越大。为了简化分析，假设 w=1，然后如果 $x_i = x'_j$，那么有 $k(x_i, x'_j) = 1$；如果 x_i 和 x'_j 相隔较远，那么 $k(x_i, x'_j) \approx 0$。换句话说，由于高斯核函数的输出衡量的是相似性，因此我们可以说，如果 x_i 和 x'_j 比较接近，那么函数这些点的均值也很接近，即 $\mu_i \sim \mu_j$。如果对 x_i 改动一点点，那么 $\boldsymbol{\mu}$ 也会相应地变化，$\boldsymbol{x}$ 的变化越大，那么 $\boldsymbol{\mu}$ 的变化也就越大。现在思考一下，我们可以将其看作是模型的一种特性，经验告诉我们，有很多函数都有类似的表现，事实上，这类函数有一个名字，称作**平滑函数**。尽管有些特例，不过许多问题都可以用平滑函数来近似。

进一步解释一下我们在做的事情，我们可以看作是在尝试使用第 1 章和第 2 章中见过的网格搜索法去拟合一个平滑的未知函数，其中网格的格点是 $\boldsymbol{x}'$，对于每个结，我们都有一个高斯函数，根据数据（通过 $\boldsymbol{\gamma}$）增加或者降低这些高斯函数的权重。如果将所有的高斯函数都加起来，那么就得到了一个平滑的曲线并拟

合出了μ。这种解释称作权重空间视角。在本章接下来的高斯过程一节中，我们将看到描述该问题的另一种视角：**函数空间视角**。

现在，将前面所有的想法汇总在一起，并应用到一个简单的合成数据集上，其中因变量是一个 sin 函数，自变量是一系列从均匀分布中得到的点。代码实现如下：

```
np.random.seed(1)
x = np.random.uniform(0, 10, size=20)
y = np.sin(x)
plt.plot(x, y, 'o')
plt.xlabel('$x$', fontsize=16)
plt.ylabel('$f(x)$', fontsize=16, rotation=0)
```



这里定义一个函数来计算模型中的高斯核函数，代码实现如下：

```
def gauss_kernel(x, n_knots=5, w=2):
    """
    Simple Gaussian radial kernel
    """
    knots = np.linspace(np.floor(x.min()), np.ceil(x.max()), n_knots)
    return np.array([np.exp(-(x-k)**2/w) for k in knots])
```

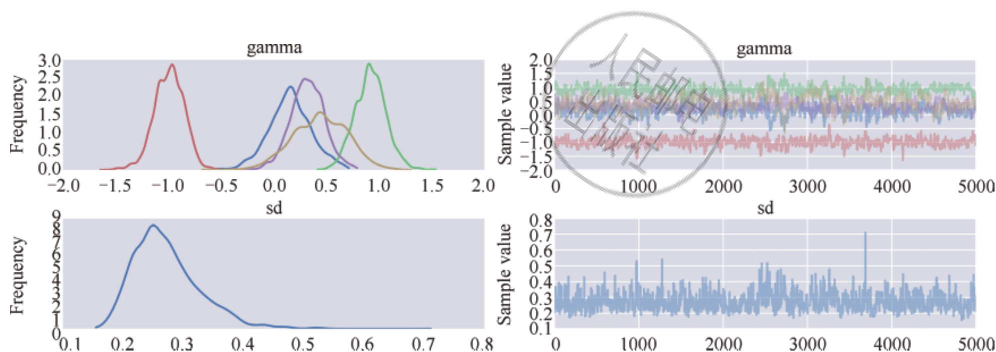
剩下的就是定义γ系数了，γ系数的个数应该与结的个数一致。注意，这里γ系数决定了估计的曲线在结这里应该增加还是降低。

有时候将模型表示成线性回归更合理，对应的数学形式如下：

$$\mu = \alpha + \beta x + \sum_{i}^{N} \gamma_i K_i(x, x')$$

不过这里我们将省略截距 α 和斜率 β，只保留最后一项。这里使用柯西分布作为先验，稍后我们会讨论在使用核函数方法过程中选择先验的一些关键点，现在先运行如下模型：

```
with pm.Model() as kernel_model:
    gamma = pm.Cauchy('gamma', alpha=0, beta=1, shape=n_knots)
    sd = pm.Uniform('sd',0,  10)
    mu =  pm.math.dot(gamma, gauss_kernel(x, n_knots))
    yl = pm.Normal('yl', mu=mu, sd=sd, observed=y)
    kernel_trace = pm.sample(10000, step=pm.Metropolis())
chain = kernel_trace[5000:]
pm.traceplot(chain);
```

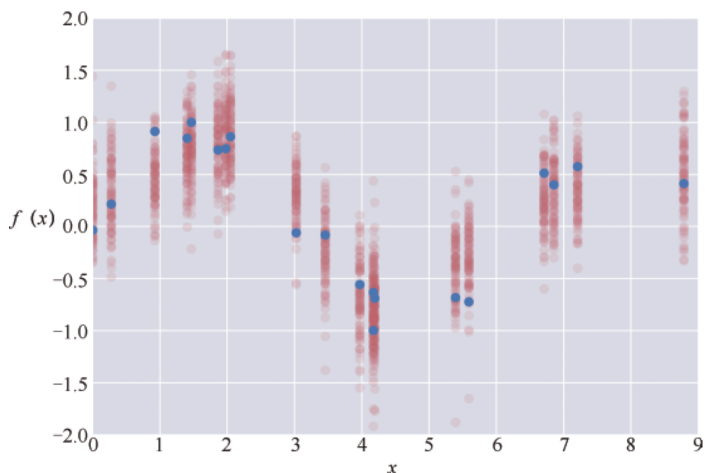


可以看到模型的结果与 sin 函数很接近，接下来对得到的模型进行后验检查，代码实现如下：

```
ppc = pm.sample_ppc(chain, model=kernel_model, samples=100)

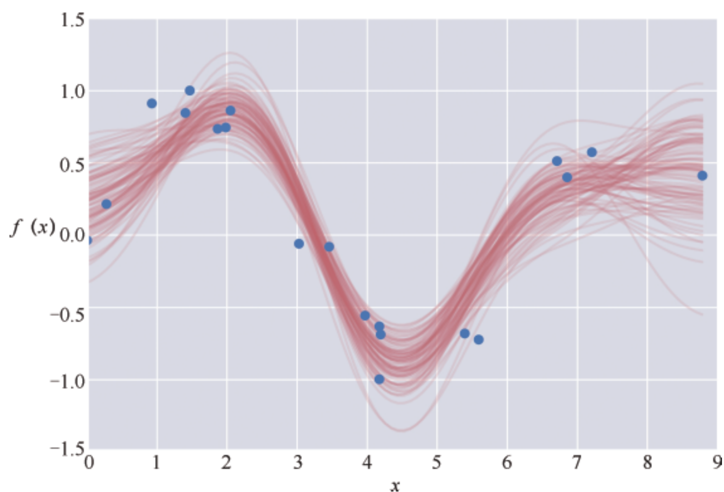
plt.plot(x, ppc['yl'].T, 'ro', alpha=0.1)

plt.plot(x, y, 'bo')
plt.xlabel('$x$', fontsize=16)
plt.ylabel(5$f(x)$', fontsize=16, rotation=0)
```



模型似乎能够很好地捕捉到数据，现在我们再用之前未使用的数据检查模型的效果，代码实现如下：

```
new_x = np.linspace(np.floor(x.min()), np.ceil(x.max()), 100)
k = gauss_kernel(new_x, n_knots)
gamma_pred = chain['gamma']
for i in range(100):
    idx = np.random.randint(0, len(gamma_pred))
    y_pred = np.math.dot(gamma_pred[idx], k)
    plt.plot(new_x, y_pred, 'r-', alpha=0.1)
plt.xlabel('$x$', fontsize=16)
plt.ylabel('$f(x)$', fontsize=16, rotation=0)
plt.plot(x, y, 'bo');
```



图中我们用蓝色的点表示数据，用红色的线表示拟合的曲线。你可能会想要修改带宽和结的个数来看看模型的效果，本章的练习 1 中有相关问题的描述，此外本章的练习 2 中还会探讨拟合另外一类函数的效果。

8.2.3 过拟合与先验

对于使用核函数方法的模型而言，人们常常关心的一点是如何选择结的个数以及位置。一种方法是直接选择数据点作为结，即将高斯分布放在数据点上，这里可以回想一下前面用过的 KDE 作图方法。

另外一种做法是，在建模过程中决定结的个数和位置。这种做法需要一些特殊的计算方法，不是很通用，或者至少不那么容易处理。

还有一种做法是应用变量选择，其思想是引入一个模型索引变量，该向量的大小与 γ 参数的大小保持一致，每个元素的值只有两种可能——0 或者 1。采用这种方法，我们可以决定模型中对应系数的开和关。不过这种方法的一个问题是，只对低维度的问题有效，因为可能的索引组合是系数个数的 2^n 倍。一种替代方案是使用正则先验，我们希望先验集中在 0 附近，从而将 γ 系数拉向 0，同时让 γ 具有长尾，从而避免拉得太厉害了。一种正则先验是柯西分布，另外一种是拉普拉斯分布。回忆一下第 6 章中我们讨论过的正则先验——岭回归和 LASSO 回归。

8.3 高斯过程

前面简要介绍了如何用核函数构建统计模型去描述任意函数。也许核函数回归听起来好像有点带有技巧性，而且其中需要指定结的个数以及分布，这似乎有点问题。现在，我们将从另外一个角度使用核函数，直接在函数空间做推断，该方法基于**高斯过程**，在数学上和计算量方面都更受欢迎。

在解释高斯过程之前，先想想什么是函数？函数可以看作是输入到输出之间的映射关系。一种学习该映射的方法是将其限定为一条直线，如我们在第 4 章做的那样，然后用贝叶斯的知识去推断出决定那条直线的参数。不过假设我

们不希望将其限定为一条直线，也可以是任意可能的函数。通常在贝叶斯统计中，对于一个不知道的量，我们先给其设置一个先验，因此，在我们不知道怎样的函数才是一个对数据拟合得很好的模型时，我们需要对函数设置一个先验。有趣的是，**多元高斯**就是这样一个先验（实际上是一个与之相似的东西，不过我们暂且这么认为），可以用一个多元高斯从很宽泛的（但很有用的）角度来描述一个函数。我们认为对于每个变量 x_i 都存在一个高斯变量 y_i，其均值和标准差暂不清楚。这样，如果向量 $\boldsymbol{x}$ 的长度为 n，那么就得到一个 n 元高斯分布。

在真实世界中，对于一个实数范围内的映射函数，$\boldsymbol{x}$ 和 $\boldsymbol{y}$ 实际上是无限多的，因为两个点之间存在着无穷多个点。因此，理论上我们需要一个无限元的高斯分布，这在数学上称为**高斯过程**，是一个参数化的**均值函数**和一个**协方差函数**，此时我们有：

$$f(\boldsymbol{x}) \sim GP(\mu(\boldsymbol{x}), K(\boldsymbol{x}, \boldsymbol{x}'))$$

关于高斯过程的一个正式定义是说，连续空间中的每个点都有一个与之对应的正态分布的变量，而高斯过程就是这无限多个随机变量的联合分布。其中均值函数是一个无限维向量的均值，协方差函数则是一个无限维的协方差矩阵，我们将看到协方差矩阵可以有效地建模 $\boldsymbol{x}$ 的变化量与 $\boldsymbol{y}$ 的变化量之间的关系。

总结一下，前面几章中，我们学习了如何估计 $p(\boldsymbol{y}\,|\,\boldsymbol{x})$，比如，在线性回归中我们假设 $y = f(\boldsymbol{x}) + \in$，其中，$f$ 是一个线性模型，然后估计线性模型的参数。也就是说，我们 $p(\theta\,|\,\boldsymbol{x})$ 是一个线性模型，并估计出了其中的参数，从而得到了 $p(f\,|\,\boldsymbol{x})$，后面将会看到，我们仍然需要估计参数，不过从概念上讲，我们是在直接处理函数，这样思考更容易理解一些。

8.3.1 构建协方差矩阵

在实践中，高斯过程的均值函数通常设为 0（尽管这并非一定的），因而高斯过程的整个行为都受协方差矩阵的控制，所以这里重点关注如何构建协方差函数。

从高斯过程先验中采样

高斯过程的概念有点像是脚手架，在实际使用中我们通常不直接使用该无限维的对象，相反，我们将无限维的高斯过程收缩到一个有限维的多元高斯。数学上，这是通过对模型中剩余的（无限的）未观测维度进行边缘化得到的，这样做之后就可以得到一个多元高斯分布。这么做遵循高斯过程的定义，即作为一系列随机变量，其中任意有限的子集都有一个联合高斯分布，于是得到的多元高斯分布的维度与我们现有数据的个数相同！这样，对于一个零均值函数的高斯过程，我们有：

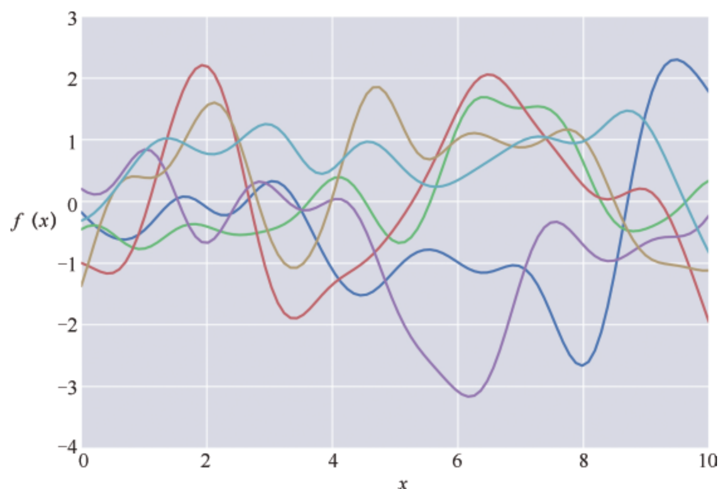
$$f(\boldsymbol{x}) \sim MvNormal(\mu=[0,\cdots,0], K(\boldsymbol{x},\boldsymbol{x}'))$$

现在我们搞定了无限维这个棘手的问题，接下来继续处理协方差矩阵，注意，我们将协方差矩阵写成了$K(\boldsymbol{x},\boldsymbol{x}')$，这里故意写成了前面核函数回归例子中的定义，因为我们实际上是用核函数在构建协方差函数。协方差函数描述的是，当 $\boldsymbol{x}$ 变化时，$\boldsymbol{y}$ 是如何随之变化的。前面核函数回归中，我们已经知道了使用高斯核函数是一个不错的假设，其等价的含义是说，一个小的 x_i 扰动会导致较小的 y_i 变化，而一个较大的 x_i 变化则会导致 y_i（平均来看）较大的变化。

为了从直观上了解高斯过程先验是什么样的，我们将对其采样并画出来。下图根据高斯过程先验画出了 6 个任意的函数（或者叫实现）。注意，这里我们将实现画成了连续函数，实际上，我们只有每个 $\boldsymbol{x}_*$ 以及与之对应的 $f(\boldsymbol{x}_*)$，不过根据平滑性假设以及无法计算出无限多个点这一事实，将实现中的点转换成连续的函数也是合理的。只要测试点足够多，对应的结果就能准确反映出函数的真实形状。

```
np.random.seed(1)
test_points = np.linspace(0, 10, 100)
cov = np.exp(-squared_distance(test_points, test_points))
  plt.plot(test_points, stats.multivariate_normal.rvs(cov=cov,
size=6).T)

plt.xlabel('$x$', fontsize=16)
plt.ylabel('$f(x)$', fontsize=16, rotation=0)
```



从这张图可以看出，高斯过程先验（以及一个高斯核函数）是一系列在 0 附近的平滑函数。

使用参数化的核函数

为了从数据中学到关于未知函数的信息，我们用一个参数化的核函数定义了协方差矩阵。我们将这里的参数称为**超参**。原因有如下两点：

- 这些参数是高斯过程的先验；
- 这个名字强调了我们使用的是非参方法。

通过学习高斯过程的超参数，我们希望拟合出未知函数。

前面我们已经提到了，核函数有很多种选择，其中最常见的是高斯核函数。前面已经看到了高斯核函数的一个版本（即带宽）。现在，我们介绍另外一个版本，用到了其他两个参数，我们可以将其写成如下形式：

$$K_{ij} = \begin{cases} \eta \exp(-\rho D) & \text{当} i \neq j \\ \eta + \sigma & \text{当} i = j \end{cases}$$

其中，D 是 SED，即$\|x - x'\|^2$；η 是一个控制垂直尺度的参数，用于让协方差矩阵建模 $f(\boldsymbol{x})$ 更大或者更小的值；参数 ρ 是带宽，前面已经知道了，ρ 控制的是函数的平滑程度；参数 σ 控制的是数据中的噪声。

让我们一起讨论下 σ 以及为什么当 i 和 j 不同时使用另外一个表达式。在一些场景中，比如进行插值的时候，我们希望模型对于每个观测值 x_i 都返回对应的观测量 $f(x_i)$ 而没有任何不确定性，而在另外一些场景中，比如本书涉及的例子中，我们希望得到 $f(x_i)$ 估计值的不确定性，因此希望得到一个接近于观测值但却又不是完全一样的值。于是有：

$$y = f(\boldsymbol{x}) + \in$$

其中，误差项通过$\in \sim N(0,\sigma)$建模。

因此协方差矩阵的构成需要考虑到有噪声的数据，于是有：

$$cov[y_i, y_j] = k(x_i, x_j) + \sigma\delta_{ij}$$

其中：

$$\delta_{ij} = \begin{cases} 0 \text{ 当} i \neq j \\ 1 \text{ 当} i = j \end{cases}$$

δ_{ij}称作 delta 分布，也就是说，我们通过将协方差矩阵的对角线上的值设置为不是正好等于 1 的值来对噪声建模，事实上，我们是直接从数据中估计出对角线上的值。这么做相当于给模型加入了一个扰动项，之所以加入该项是因为，在核函数所附带的假设中，两个输入越接近，对应的输出也越接近，极限情况下，两个输入完全相等，它们的输出也应该完全相等，而增加扰动项则有利于捕捉到观测值的不确定性。

为了更好地理解超参的含义，我们可以把扩展后的高斯核函数画出来，你也可以随意选一些不同的超参值自己试试，代码实现如下：

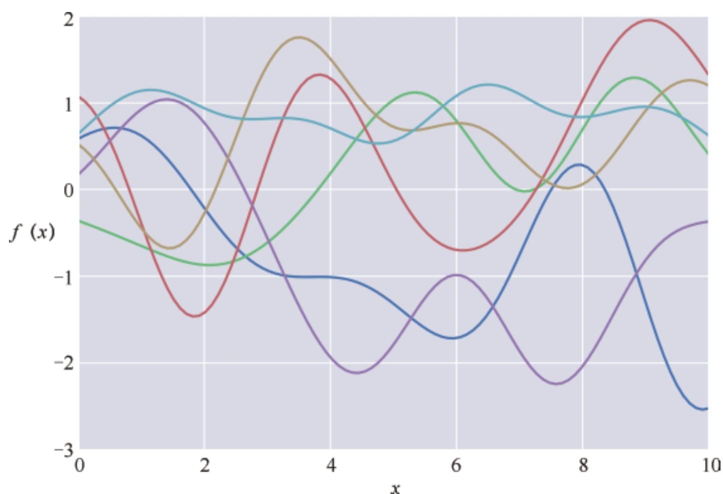
```
np.random.seed(1)
eta = 1.5
rho = 0.2
sigma = 0.007

D = squared_distance(test_points, test_points)

cov = eta * np.exp(-rho * D)
diag = eta + sigma

np.fill_diagonal(cov, diag)
```

```
for i in range(6):
    plt.plot(test_points, stats.multivariate_normal.rvs(cov=cov))
plt.xlabel('$x$', fontsize=16)
plt.ylabel('$f(x)$', fontsize=16, rotation=0)
```



8.3.2 根据高斯过程做预测

接下来学习如何用高斯过程做预测。高斯过程的一个优点就是结果可以从数学分析上直接推导。如果将高斯过程先验与高斯似然组合在一起，我们得到的是一个高斯过程后验。也就是说，如果在一些测试点上应用条件高斯，我们可以得到如下后验预测密度的表达式：

$$p(f(\boldsymbol{X}_*) \mid \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y}) \sim N(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$$

$$\mu = \boldsymbol{K}_*^T \boldsymbol{K}^{-1} y$$

$$\boldsymbol{\Sigma} = \boldsymbol{K}_{**} - \boldsymbol{K}_*^T \boldsymbol{K}^{-1} \boldsymbol{K}_*$$

在给定数据 $\boldsymbol{X}$、$\boldsymbol{y}$ 以及一些测试点 $\boldsymbol{x}_*$ 的条件下，计算未知函数在（暂时）未知的点上的值 $f(\boldsymbol{x}_*)$。注意，这里我们使用符号 * 来表示所有测试点上的运算，其中：

$$\boldsymbol{K} = K(\boldsymbol{X}, \boldsymbol{X})$$

$$\boldsymbol{K}_{**} = K(\boldsymbol{X}_*, \boldsymbol{X}_*)$$
$$\boldsymbol{K}_* = K(\boldsymbol{X}, \boldsymbol{X}_*)$$

这个式子初看可能有点吓人，稍后我们将看到如何用代码描述上面的式子（但愿这样理解起来清晰一些），不过现在我们先从直观上理解这些式子。

K_{**} 是 $\boldsymbol{x}_*$ 自己的协方差，其实就是 $\boldsymbol{x}_*$ 的方差。也就是说，测试点的方差其实就是先验的方差，注意，这里Σ等价于 K_{**} 减去一个量，这个式子的含义是说，我们可以根据数据将先验的方差降低这么多。此外，对于一个距离数据点 x_i 较远的测试点 $\boldsymbol{x}_*$ 而言，核函数返回的值接近于 0，因而这个量也接近于 0，结果就是先验的方差也不会降低多少。换句话说，评估远离数据点的函数并不会降低不确定性。因此，推断基于的是数据在测试点附近的局部特性。

如果你想知道更多有关多元高斯的数学特性及前面的表达式是如何推导的，可以参考本章的阅读更多部分，我在那里添加了一些你可能感兴趣的参考资料。针对当前的讨论内容，我们暂且假设前面的后验预测的表达式是已知的。有一点需要注意的是，我们可以从高斯过程后验中进行采样。我们得到的这个描述未知函数的表达式非常有用，不过有一个问题是，计算过程中涉及求解矩阵的逆，而计算逆矩阵的复杂度是 $O(n^3)$，如果你不知道这个表达式的含义，可以理解为该操作很慢，因此实际使用中，我们无法将高斯过程应用到超过几千个数据点的场景，不过对于这种情况有一些近似的方法来加速计算，这里暂不深入讨论。此外，在实践中，直接求逆矩阵可能会有一些数值问题，而且不稳定，因此更倾向于使用一些替代方案，比如用 **Cholesky 分解**来计算均值和协方差后验函数。Cholesky 分解有点像我们熟悉的对标量求平方根，不过其对象是矩阵。

在深入 Cholesky 分解和直接求逆矩阵的例子之前，我们先观察下。如果求逆矩阵会有问题，那么为什么还要将高斯过程表示成协方差矩阵而不是直接用协方差矩阵的逆呢？原因是，当我们使用协方差矩阵的时候，对其子集的计算是独立于其余部分的计算的，因而它是独立于未观测到的点的计算。不过对于逆协方差矩阵而言，数据子集的计算会依赖于我们是否观测到了其余点。只有使用协方差矩阵（而不是它的逆），我们才能得出高斯过程基于一系列随机变

量一致的定义。

总结一下，为了完全从贝叶斯的角度使用高斯过程去近似一个函数，我们需要：

- 选择一个核函数构建一个多元分布的协方差矩阵；
- 用贝叶斯统计的方法推断出核函数的参数；
- 计算出每个测试点的均值和标准差。

注意，实际上我们并没有真正地计算出高斯过程，只是借用了这个数学概念来确保我们所做的是合理的，在实践中，所有的计算都是通过多元高斯完成的。

读完前面理论方面的长篇大论之后，终于到了大家期待的时刻，我们将把这些想法转化成代码。首先，假设已经知道了核函数的参数，然后用代码描述后验的表达式。用到的超参和 test_points 的值与前面定义的相同，数据也与前面核函数回归例子中用到的一样，代码实现如下：

```
np.random.seed(1)

K_oo = eta * np.exp(-rho * D)

D_x = squared_distance(x, x)
K = eta * np.exp(-rho * D_x)
diag_x = eta + sigma
np.fill_diagonal(K, diag_x)

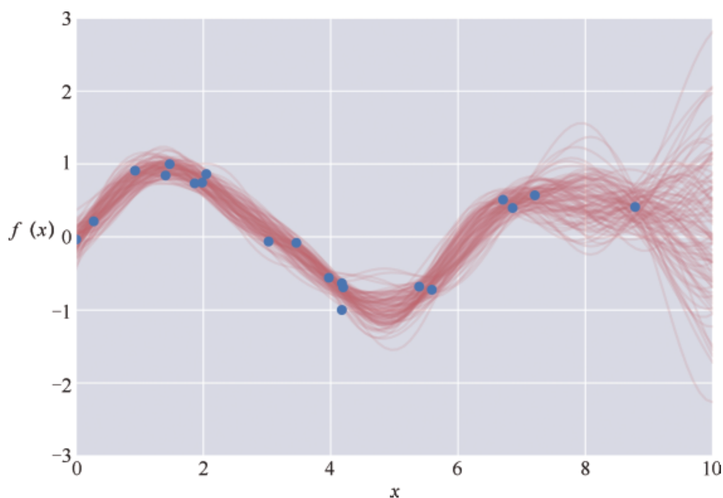
D_off_diag = squared_distance(x, test_points)
K_o = eta * np.exp(-rho * D_off_diag)

mu_post = np.math.dot(np.math.dot(K_o, np.linalg.inv(K)), y)
SIGMA_post = K_oo - np.math.dot(np.math.dot(K_o, np.linalg.inv(K)), K_o.T)

for i in range(100):
    fx = stats.multivariate_normal.rvs(mean=mu_post, cov=SIGMA_post)
    plt.plot(test_points, fx, 'r-', alpha=0.1)

plt.plot(x, y, 'o')

plt.xlabel('$x$', fontsize=16)
plt.ylabel('$f(x)$', fontsize=16, rotation=0)
```



这里通过叠在一起的一些红线（从高斯过程中得到的实例）来表示不确定性。从图中可以看到，在数据点附近的不确定性要小一些，而在最后两个数据点之间的不确定性要大一些，在最右边没有数据点的地方（$x \sim 9$）不确定性要更大一些。

接下来，我们将重新实现前面的计算过程，不过这次用的是 Cholesky 分解。下面的代码是根据 Nando de Freitas 讲授机器学习课程中的实例修改后得到的，在代码中，N 表示数据点的个数，n 表示测试点的个数。

```
np.random.seed(1)
eta = 1
rho = 0.5
sigma = 0.03

f = lambda x: np.sin(x).flatten()

def kernel(a, b):
    """ GP squared exponential kernel """
    sqdist = np.sum(a**2,1).reshape(-1,1) + np.sum(b**2,1) - 2*np.dot(a, b.T)
    return eta * np.exp(- rho * sqdist)

N = 20
n = 100

X = np.random.uniform(0, 10, size=(N,1))
y = f(X) + sigma * np.random.randn(N)

K = kernel(X, X)
```

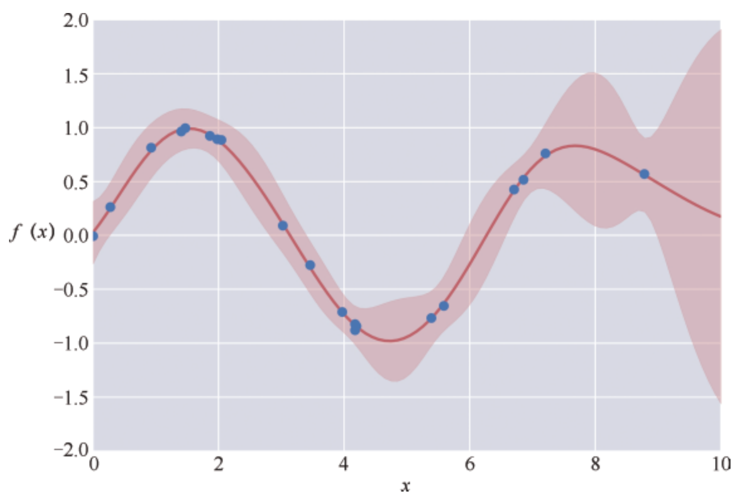
```
L = np.linalg.cholesky(K + sigma * np.eye(N))

test_points = np.linspace(0, 10, n).reshape(-1,1)

Lk = np.linalg.solve(L, kernel(X, test_points))
mu = np.dot(Lk.T, np.linalg.solve(L, y))

K_ = kernel(Xtest, Xtest)
sd_pred = (np.diag(K_) - np.sum(Lk**2, axis=0))**0.5

plt.fill_between(test_points.flat, mu-2*s, mu+2*s, color="r", alpha=0.2)
plt.plot(test_points, mu, 'r', lw=2)
plt.plot(x, y, 'o')
plt.xlabel('$x$', fontsize=16)
plt.ylabel('$f(x)$', fontsize=16, rotation=0)
```



这里我们将数据点用蓝色的点来表示，均值函数是一条红线，不确定性用半透明的区域来表示。

8.3.3　用 PyMC3 实现高斯过程

总结一下，我们有如下高斯先验：

$$f(x) \sim GP(\mu = [0, \cdots, 0], k(x, x'))$$

一个高斯似然：

$$p(y \mid x, f(x)) \sim N(\boldsymbol{f}, \sigma^2 I)$$

以及一个高斯过程后验：

$$p(f(\boldsymbol{x}) \mid \boldsymbol{x}, \boldsymbol{y}) \sim GP(\mu_{post}, \Sigma_{post})$$

记住，在实践中，我们使用的是多元高斯，因为在一个有限的数据集上，高斯过程就是一个多元高斯。

我们将使用贝叶斯相关的原理来学习协方差矩阵的超参。你会看到，尽管使用 PyMC3 很简单，但是现在代码量会增加一些，我们需要手动求解矩阵的逆（或者计算 Cholesky 分解）。

很可能在不久的未来，PyMC3 中会出现一个高斯过程模块，让高斯过程模型的构造更简单一些。说不定在你阅读本书的此刻，高斯过程模型已经有了！[①]

下面的模型是从 Chris Fonnesbeck 写的 Stan 代码中改造的：


```
with pm.Model() as GP:
    mu = np.zeros(N)
    eta = pm.HalfCauchy('eta', 5)
    rho = pm.HalfCauchy('rho', 5)
    sigma = pm.HalfCauchy('sigma', 5)

    D = squared_distance(x, x)

    K = tt.fill_diagonal(eta * pm.math.exp(-rho * D), eta + sigma)

    obs = pm.MvNormal('obs', mu, tt.nlinalg.matrix_inverse(K), observed=y)

    test_points = np.linspace(0, 10, 100)
    D_pred = squared_distance(test_points, test_points)
    D_off_diag = squared_distance(x, test_points)

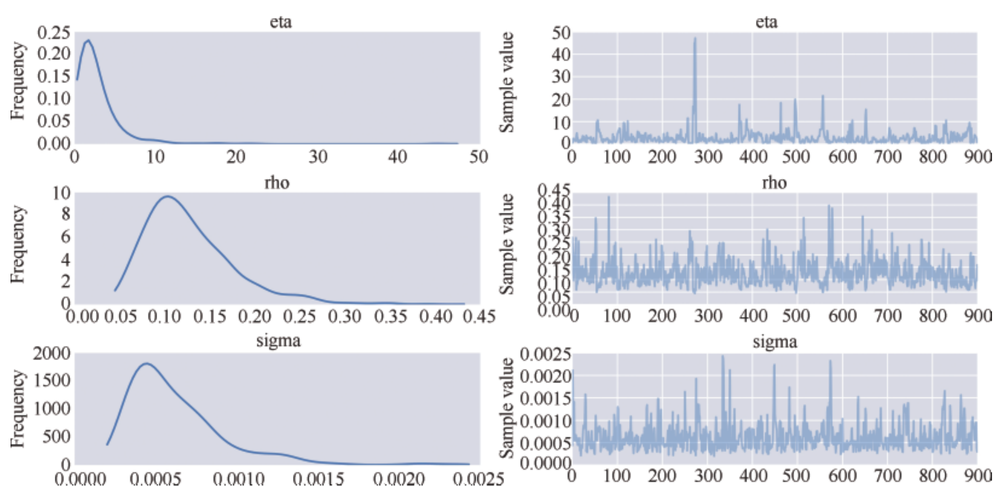
    K_oo = eta * pm.math.exp(-rho * D_pred)
    K_o = eta * pm.math.exp(-rho * D_off_diag)
```

① 在最新版的 PyMC3 中，已经有该模块了，具体请参考 https://pymc-devs.github.io/ pymc3/examples.html#gaussian-processes。——译者注

```
        mu_post = pm.Deterministic('mu_post', pm.math.dot(pm.math.dot(K_o,
tt.nlinalg.matrix_inverse(K)), y))
        SIGMA_post = pm.Deterministic('SIGMA_post', K_oo - pm.math.dot(pm.
math.dot(K_o, tt.nlinalg.matrix_inverse(K)), K_o.T))

        start = pm.find_MAP()
        trace = pm.sample(1000, start=start)

    varnames = ['eta', 'rho', 'sigma']
    chain = trace[100:]
    pm.traceplot(chain, varnames)
```



如果留心的话，你会注意到估计出来的参数（η,ρ,σ）的均值就是我们前面例子中用到的值。这也解释了为什么拟合的效果这么好，这些超参可不是从我们脑袋里想出来的！

```
    pm.df_summary(chain, varnames).round(4)
```

	mean	sd	mc_error	hpd_2.5	hpd_97.5
eta	2.5798	2.5296	0.1587	0.1757	6.3445
rho	0.1288	0.0485	0.0027	0.0589	0.2290
sigma	0.0006	0.0003	0.0000	0.0002	0.0012

后验预测检查

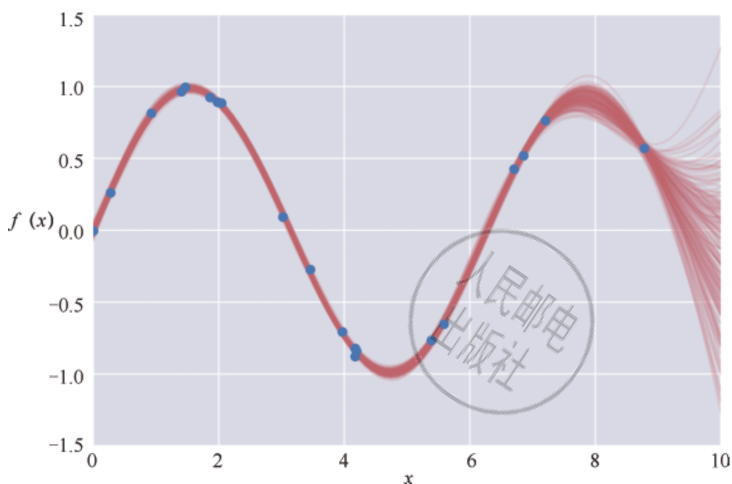
现在我们把原始数据和从高斯过程后验中得到的实例都画出来，注意，我们

还画出了超参的不确定性，而不只是它们的均值，代码实现如下：

```
y_pred = [np.random.multivariate_normal(m, S) for m,S in zip(chain['mu_post'][::5], chain['SIGMA_post'][::5])]

for yp in y_pred:
    plt.plot(test_points, yp, 'r-', alpha=0.1)

plt.plot(x, y, 'bo')
plt.xlabel('$x$', fontsize=16)
plt.ylabel('$f(x)$', fontsize=16, rotation=0)
```



周期核函数

前一幅图中，你可能注意到了，我们可以很好地拟合 sin 函数，不过模型在 9 到 10 之间不确定性非常大（该区间没有数据点）。该模型的一个问题是，原始的数据是通过一个周期函数生成的，但是我们的核函数并没有做出周期性的假设。某些情况下，当我们知道数据可能是周期性的时候，应该使用一个周期性的核函数。一个典型的周期性核函数如下：

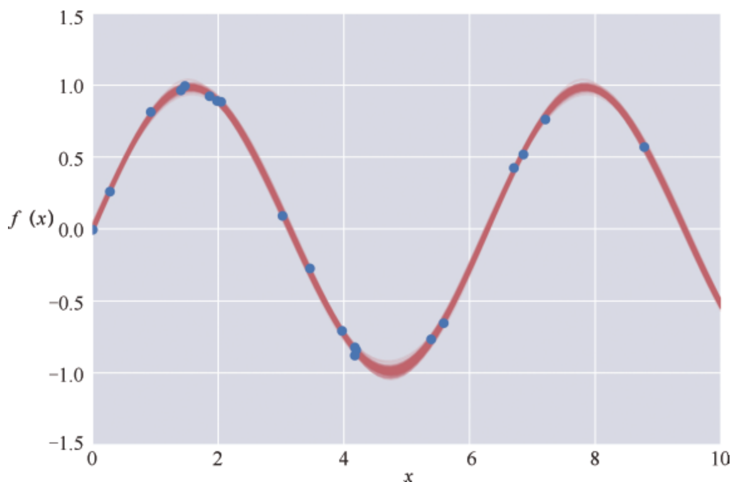
$$Kp(\boldsymbol{x},\boldsymbol{x}') = \exp\left(-\frac{\sin^2\left(\frac{\boldsymbol{x}-\boldsymbol{x}'}{2}\right)}{w}\right)$$

注意，这个核函数与高斯核函数的主要区别是包含了一个 sin 函数。我们可以复用前面的代码，唯一的区别是现在需要定义一个周期函数而不是 squared_

distance，代码实现如下：

```
    periodic = lambda x, y: np.array([[np.sin((x[i] - y[j])/2)**2 for i in
range(len(x))] for j in range(len(y))])
```

在这个模型中，我们需要将 squared_distance 替换成这个周期函数。运行模型之后，你应该会得到类似下面的图：



8.4 总结

本章一开始，我们学习了贝叶斯框架下的非参统计，以及如何用核函数表示统计学中的问题，例如，我们用一种采用核函数的线性回归去建模了非线性输出，随后继续讨论了另外一种构建和理解核函数方法的方式——高斯过程。

高斯过程是多元高斯分布扩展到无限多维时的一种一般形式，可以用一个均值函数和一个协方差函数来描述。由于从概念上我们可以将函数看作无限长的向量，因而可以将高斯过程作为函数先验。实践中，我们处理的是维度和数据点个数相同的多元高斯分布。为了定义与之对应的协方差函数，我们使用了参数化的核函数，通过学习超参数，最终可以拟合出任意复杂的未知函数。

这一章中，我们简要介绍了高斯过程，还有许多与之相关的主题需要学习（比如构建一个半参数化模型，将线性模型作为均值函数），或者是将两个或者多个核函数组合在一起来描述未知函数，或者是如何将高斯过程用于分类任务，

或者是如何将高斯过程与统计学或者机器学习中的其他模型联系起来。不管怎么说，我希望本章对高斯过程的介绍以及本书中一些其他主题的介绍能够激励你阅读、使用和进一步学习贝叶斯统计。

8.5　深入阅读

- Carl Edward Rasmussen 和 Christopher K. I. Williams 写的《Gaussian Processes for Machine Learning》一书。
- Kevin Murhpy 的《Machine Learning a Probabilistic Perspective》中的第 4 章和第 15 章。
- 《Statistical Rethinking》中的第 11 章。
- 《Bayesian Data Analysis, Third Edition》中的第 22 章。

8.6　练习

1．在核回归的例子中，尝试修改结的个数以及带宽（一次修改一个），这些改变有什么效果？尝试只使用一个结，你观察到了什么？

2．用核回归拟合其他函数，比如 $y = np.\sin(x) + x{**}0.7$ 或者 $y = x$。尝试像练习 1 中那样修改数据和参数的个数。

3．在前面从高斯过程先验中采样的例子里，增加实例的个数，将 `plt.plot(test_points, stats.multivariate_normal.rvs(cov=cov, size=6).T)` 替换成 `plt.plot(test_points, stats.multivariate_normal.rvs(cov=cov, size=1000).T, alpha=0.05, color='b')`。高斯过程的先验是什么样子的？你是否看出 $f(x)$ 分布得像均值为 0、标准差为 1 的高斯分布？

4．对于一个使用高斯核的高斯过程后验，尝试将测试点定义在区间 [0,10] 之外，区间外的点有怎样的结果？这告诉我们在外推（extrapolating）时需要注意什么（特别是非线性函数）？

5．重复练习 4，这次换成周期性核，现在你的结论又是什么？

本书介绍了贝叶斯统计中的主要概念，以及将其应用于数据分析的方法。本书不要求读者有任何统计学方面的基础知识，不过需要读者有使用Python的经验。本书采用编程计算的实用方法介绍了贝叶斯建模的基础，使用一些手工构造的数据和一部分简单的真实数据来解释和探索贝叶斯框架中的核心概念，然后在本书涉及的模型中，抽象出了线性模型用于解决回归和分类问题，此外还详细解释了混合模型和分层模型，并单独用一章讨论了如何做模型选择，最后还简单介绍了非参模型和高斯过程。

本书所有的贝叶斯模型都用PyMC3实现。PyMC3是一个用于概率编程的Python库，其许多特性都在书中有介绍。在本书和PyMC3的帮助下，读者将学会实现、检查和扩展贝叶斯统计模型，从而解决一系列数据分析的问题。

从本书你将学到：

- 从实用的角度理解基本的贝叶斯概念；
- 学习如何用PyMC3构建概率模型；
- 掌握检查和修改模型的技能；
- 利用分层模型的优势给模型加入结构；
- 针对不同的数据分析问题，找到合适的模型；
- 学会在不确定的情况下做模型选择；
- 用回归分析预测连续变量，用逻辑回归或softmax做分类；
- 学习如何从概率的角度思考，释放贝叶斯框架的灵活性与力量。





异步社区 www.epubit.com.cn
新浪微博 @人邮异步社区
投稿/反馈邮箱 contact@epubit.com.cn



分类建议：计算机／程序设计／Python
人民邮电出版社网址：www.ptpress.com.cn